Robot Games: translating game interaction to robotic interface

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Michael David Janssen

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Nikolaos Papanikolopoulos

August, 2014

Acknowledgements

The work in this thesis was supported in part by the National Science Foundation through grants #IIP-0934327, #IIP-1032018, #IIS-1017344, #CNS-1061489, #CNS-1138020, #IIP-1127938, and #IIP-1237259.

The author would like to thank his advisor, Dr. Nikolaos Papanikolopoulos for his advice and support for many years. The author also thanks his wife, Diana Rajchel, for her endless support, feedback, sane-keeping, and advice. The author extends a special thanks to Andrew Drenner for his significant support, proofreading of many versions of this document, helpful discussion, and suggestions. The author also thanks his colleagues, William Beksi, Robert Bodor, Nate Bird, Ian Burt, Dario Canelon, Duc Fehr, Brett Hemes, Alex Kossett, Apostolas Kottas, and Ben Miller for their ideas, support, and feedback.

Dedication

To my dad, who started me down this path so many years ago.

Abstract

The video game industry, while refining hardware, techniques, graphics technology, and software, produce a variety of interfaces for controlling and representing the virtual worlds they present. This feedback and controls convey control of a world analogous to the real world, with interactions helping the player effect that world. These methods used in games can be translated usefully for controlling robotic platforms in the real world. The amount of robots used in the field for military, commercial, and disaster response increases every year. Using games as inspiration, these robot systems can be improved to produce more efficient interfaces, utilize mass-produced commodity hardware, and reduce training time with operators increasingly familiar with these types of control.

This dissertation proposes a method for transferring game interactions to robotic interfaces. The method works in five stages: identifying the task as a common robot task, choosing a game interaction and describing it, translating elements to robotic interface requirements, developing interfaces using these mappings, and testing the resulting interfaces to determine the impact of the changes. To help researchers use the method, a survey of common game interactions are identified from recent popular games and described for the second stage of the method.

A framework for remote robot interaction studies was developed and is presented, using a client-server architecture enable participants in a robot interface user study to simulate robots and submit study results. This framework provides many advantages over a traditional in-person user study.

Four robot interfaces are developed using the method, tested using a variety of evaluation methods. The first uses GPS and planning to command multiple robots, tested through experimental trials and three scenarios. The second studies selection and formation movement techniques when interacting with multiple robots, using the remote study framework for testing. The third also uses the remote study framework, focusing on adding a queue of future actions to a supervisory control interface. The last interface uses augmented reality in a teleoperation interface to show a virtual trail

of previously visited locations. This interface was validated through a usability study with a mobile robot exploration task, showing significant improvement in efficiency.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Whether for use in the military, for commercial applications, disaster response, or in the home, use of mobile service robots is increasing. In 2012, more than 16,000 service robots were sold[1], with many being Unmanned Aerial Vehicles or Unmanned Ground Vehicles for use in military applications. While the military is the largest user of these robots, domestic and professional robot use of mobile robotic systems is also increasing. Similar to the proliferation of computers in the latter half of the 20th century, robots are not being used in increasingly diverse fields. These robots will necessarily be exposed to more robot operators across these varied uses. The increasing numbers and penetration of mobile robot platforms means a focus must be placed on how one or more robots can be operated efficiently with less training.

While increasingly sophisticated methods of automation mean more complex actions are available for operated robots, humans will always need to be in the loop to provide direction. They will either be directing the robots partially or completely, or providing assistance in tasks that humans excel and robots do not, such as object of interest recognition and high-level decision making. Human-Robot Interaction is the field concerned with human interaction with any robot, including stationary arm robots, robots sharing work areas with humans, or ones primarily interacting as guides or co-working assistants. Human-Robot Interaction researchers continue to improve methods of communication and interfaces to increase the effectiveness and efficiency of robot operators. The focus of this thesis is a subset of this field, to improve remote mobile robot control.

Mobile robot control typically operates in one of three categories today: highly

automated, supervisory control, and teleoperation. These three categories provide robot control through very different interfaces. Highly automated systems' interfaces provide for minimal interaction, with some targeting no interaction at all, only including an operator when necessary. Supervisory control and teleoperation robots differ as both require an operator and use a graphical interface to control a robot or team of robots at a distance. The increasing abilities of mobile robotic systems are result in a trend towards interfaces that are more complex to use these abilities. Novel interaction models is one way to reduce or mitigate this newly introduced complexity.

Video games are one place where these interactions could be found. Tasks that are being performed by the players of modern games can be related closely to the tasks which are required of the mobile robotic systems. The players of these games are using a screen-based interface to control the avatar. They are physically separated from the units which have mobility and a large set of abilities that must be activated to achieve the goals set out by the game narrative. It is not hard to imagine that an interface similar to these game interfaces could therefore be used to control a robotic system.

Video games also have a number of advantages beyond the similarities to the mobile robot task. They are a very popular form of entertainment, with more than half of the American population playing[2], meaning game interactions are familiar to many people. Games take a complex set of actions and provide an interface that is simple to learn and easy to use. The market for games is large and highly competitive - games with controls which are "awkward" or "hard to use" are seen as a major detriment to enjoyment of a game, producing negative reviews[3, 4]. These motivations to make the game easy to use and the popularity of games could make it easier to find the future operators of robots, and reduce or eliminate the training time required for the systems.

Game designers also want to reduce this training time within their games, making their games easier to learn and be proficient in. At the same time, they balance this simplification with the desire for the player to feel have accomplished something through their skill. One way for a designer to achieve both goals this is to reuse an interface or controls which are already familiar to the player. Thus, similar types of games share basic control schemes with others in their genre. These common interaction elements are already being generalized, further homogenizing the skills that would be used in a game-like interface. This common language among designers and players of games encourages

learning skills transferable to other games, and provides for a better experience for both.

These advantages of video games provide a meaningful reason for the study of game interactions for improving the robotic interface. Examining the interaction of a player within a recent popular game such as Battlefield 4[5], movements and controls that the player provides to move their avatar and view may be mapped closely to an interaction between a robot operator and a teleoperated robot. The ease of control and level of awareness of the environment that is provided the game player is not matched by the interfaces used for controlling a teleoperated robot. Motivated by a desire to improve mobile robotic control interaction, and with this association in mind, the main thesis is stated:

> ***Interaction techniques in common use in video games can translate to valuable improvements for mobile robotics control interfaces.***

Using these techniques for to improve robot interfaces, the overarching objective of the work is to create interfaces which are easier to use by the general public. For new operators who are already players of games, using these game interaction techniques will provide a familiar way for them to control robots. Even those not familiar with games will benefit as game interactions have been tuned by years of research to provide the easiest to learn and use, making the resulting interfaces easy to use.

1.1 Method

Human-Robot interaction designers are not typically familiar with game interfaces. To study the use of game interactions within robotic interfaces, a framework to add these elements is necessary. The interaction between the game player and the game itself needs to be defined, and their relationship to the robot operator interacting with robots should be elaborated. Game interfaces must also be studied to find the beneficial interactions, which can be used by the designers of interfaces. Finally, these new interfaces should be studied using metrics that are accepted by the community to show the advantages of the game elements.

When considering game and robot control, four major entities are involved. For

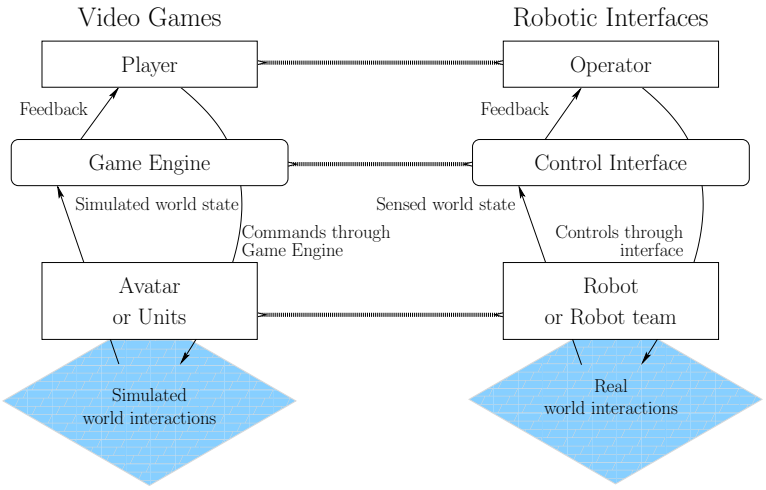


Figure 1.1: Gaming correlations to robot control.

clarity, the person who is interacting with a game is referred to as a **player**. A player of a game interacts with the game to accomplish their goal, which is either provided by the game ("defeat the aliens"), or internally produced ("get a faster time"). The player uses an **input method** to interact with the game, which the **game engine** uses to affect the game world. The game engine then provides the player with **feedback** based on the actions taken and new state of the game world. Often the player will be affect the game world through a explicit or implicit character in the world which is directly controlled by most inputs provided. This controlled character is referred to as the player's **avatar**. If the player instead issues commands to a set of characters, with most inputs related to the management of those characters, they can instead referred to as the player's **controlled units**.

As noted earlier, there is a compelling correlation between robot interaction and video game playing as illustrated in Figure 1.1. Considering these correlations, focus is placed on tasks that are successful in the realm of games which are analogous to the robotic control realm, as these are the most likely to be transferred successfully. Using the methods provided by games to cause effect in a simulated world, interfaces can be developed to interact with the real world to produce an expected effect.

The player, avatar, and controlled units have their counterparts within a robotic interface. The person interacting with a robotic interface uses an **input method**, but

is now referred to as the **operator**. The operator interacts using a **control interface** to affect the real world remotely. That interface controls a remote **robot**. The robot sensors to provide information to the control interface, which provides the player with **feedback** based on them. If multiple robots are being controlled simultaneously, they make up a **team of robots** instead. These terms are used to refer to the robot or robots being controlled implicitly and not other robots which may also be included. Robot operators are operating the robot for a **mission** which is made up of a number of **tasks**. This mission is usually provided separately, although new tasks can be generated by circumstances.

The framework developed helps exploit this correlation to improve robotic interfaces. A high-level flowchart of the framework is shown in Figure 1.2. The framework starts with the desired robot task, capabilities of the robotic system being used, and the mission parameters if known. Initially these parameters need to be decomposed into a categorization of tasks that can be used to choose beneficial interactions.

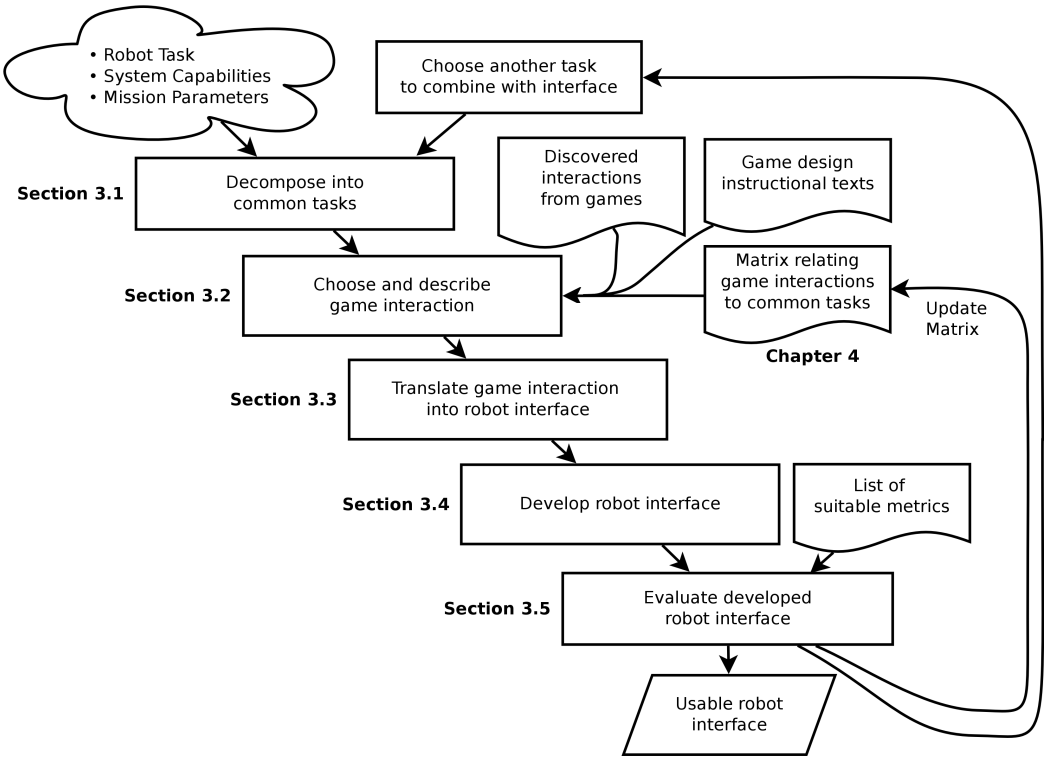


Figure 1.2: Basic flow of the framework

Thousands of games over the long history of video games have provided a surfeit of interactions to choose from. The robot task is used to filter these interactions, which are chosen using one of three methods. The primary method is a matrix of interactions which is built from studying and experimenting with the framework. An initial matrix is researched and presented in Chapter 4. Interactions can also be found thorough interface descriptions in game design texts, or discovered by examining games for interactions. By focusing first on the task matrix filtering the interactions, a strong analogy between the game player and the robot operator can be maintained.

Once the task is identified and the game interaction to be implemented has been chosen, it needs to be transferred to the robotic system. The interaction is merged by identifying key elements of the game interaction and mapping them to their analogous real-world components. This mapping is then used to produce an enhanced robotic interface implementing a similar interaction on the robotic system.

Lastly, the interfaces produced using this method must prove to be better for the tasks they are designed to improve. To test the interface, studies are performed using both objective and subjective metrics chosen to measure effectiveness and efficiency of at the task. These studies can be experimental, user studies, or comparative studies.

A method for performing structured remote user studies for robotics was developed. This method introduces a number of advantages including a larger participant pool, asynchronous participation and automated data collation. The robot system is simulated using research-level libraries on the participant's computer and combined with monitoring to provide a set of participant data to evaluate the interfaces. Experiment results are submitted and study progress is monitored, analyzed and promoted using this cloud-based study framework.

Using this method, a robotics researcher can produce interfaces which are easier to use and outperform current interfaces on task-focused metrics. The information learned from the experimentation can be integrated into the matrix of interactions, which can be used by other interface designers to enhance their interfaces for similar tasks on different robotic systems. This framework can also be reused iteratively to continue to improve the interface if desired. Further, the framework adapts to incorporate future game interactions, leveraging research in game development for the similar task of robotics control.

Four interfaces are used to demonstrate the principles of the developed framework from the design stage through experimentation. Three focus on supervisory robot control methods. The first is a map-based interaction with a heterogeneous set of robots used as target robots and experimental trials of the developed interface were performed. The most recent interface develops an interaction into a teleoperation and exhibits many of the benefits, including building on game technologies and incorporating game-like control.

Two interfaces which utilize simulation to implement different game interaction methods are presented, each of which are validated by a remote user study. These interface experiments use the remote structured study method which is administered using participants' computers, showing the benefits of using the method. These experiments demonstrate both the framework for game interaction and the remote user study software for the development of robotic interfaces.

1.2 Contributions

In this thesis, three key contributions are presented:

1. *Framework for translating game interactive elements to robotics* - A framework for developing improvements to robotic interfaces is described, providing the method to transfer game interactions for use in robotic interfaces. The framework uses a five-step process (as seen in Figure 1.2) to develop portable robot interface descriptions from game interactions. A survey of game interactions sourced from recent popular games provides an initial set of interactions and common behaviors to implement, provided as a reference and adaptable matrix (Table 4.3).

2. *Development of structured remote user studies for robotics* - A method for conducting robotic user studies remotely using simulated robots, providing a number of benefits including shorter participant time required, easy collation of results including study progress, increased data gathering ability, and a relatively shorter time period required for testing.

3. *Novel user interface for exploration and mapping* - the final user interface represents a novel interface enhancing exploration of unknown environments by marking

explored territory using augmented reality methods, with lower sensing requirements than traditional mapping methods.

1.3 Structure

The rest of this thesis is as follows:

- Chapter 2 reviews similar work in related fields, framing the contributions while differentiating from them. Significant papers defining the task categories for filtering, as well as sources for metrics used in the framework are also introduced.
- Chapter 3 provides full details on all steps of the Game Interaction to Robotics framework. Motivation for procedures of the framework are explained, along with the different requirements for each stage and the contraindications that can occur at each step of the process. This chapter also contains the menu of metrics for the different task types. An example task is used to demonstrate the various stages of the framework.
- In Chapter 4, a survey of modern popular games is performed to discover common interactions in use applicable to the common robotic tasks. Approximately thirty interactions were discovered and are organized with relation to the common tasks. Each interaction's source is described with benefits to the task, a sample parameterization and some mappings to sample robotic interfaces are presented. This chapter provides the initial matrix to use in the second step of the framework.
- Chapter 5 contains an in-depth look at the structured remote user study for robotics as developed. The advantages and pitfalls of using such a framework are communicated along with the various options for researchers to customize the studies and results gathering.
- Chapter 6 introduces the game interfaces developed using game interactions and connects their development to the various game interactions through the matrix, outlaying benefits provided using game interactions in comparison to other methods of implementing the same interfaces.

- Each interface, along with the design and results of experiments performed using that interface are presented in detail:
  - SMuRC, a GPS-enabled robotic guidance interface using a top-down interface for remote supervision of multiple robots is discussed in Chapter 7.
  - A multi-robot teaming interface is developed in Chapter 8, testing interactions for formation and group selection. Inspired from game interactions, the methods focus on management and navigation. A comparative study is run using an early version of the structured remote user study.
  - Chapter 9 presents the development of a multi-robot interface improvement adding queued actions to improve efficiency when managing many robots. The experimental method uses the more developed version of the structured remote user study.
  - In Chapter 10 a novel interface improvement for teleoperative exploration of unknown environments is developed. The framework is used to full effect to integrate the interaction and guide the evaluations. A user study showing the effect of the interface improvement was conducted, and results are presented. The interface uses augmented reality methods to mark already explored territory in an intuitive manner.
- Chapter 11 revisits the contributions in relation to the main thesis, and explores possible directions for future work.

Chapter 2

Related Work

Research related to the framework presented here originates from many fields of study. Related work from the fields of video game study, human-computer interaction, and human-robot interaction are discussed here. Context differentiating the previous work from the goal and contribution of the framework is presented when applicable. Valuable work which either reinforces the motivations, referenced and/or used in the framework is also placed in relation to the thesis. Video games are the source media for interface improvements, and game studies related to interface elements and interaction methods help motivate and provide valuable input on current game interaction design. The field of human-computer interaction has a number of studies related to games which provide reinforcement for the types of interface changes which are suggested by the framework, and draw a critical eye on game interfaces. Human-robot interaction has provided a number of interface metrics and guidelines which should be considered. A number of previous interfaces either are explicitly influenced by games, or present suitably similar interaction elements which should be considered as earlier work. Work covered in this chapter relates to the development of the game interaction to robotics framework. Other work is referenced in relation to the remote robot interface user study method proposed, and the developed robot control interfaces. These are covered in sections of their respective chapters.

2.1 Video Game Studies

Video games are a popular form of entertainment, with more than half of the American population playing, with revenues over 20 billion dollars yearly[2]. Partially due to popularity of the medium, video games and game players have been researched for some time. With some looking back to foundational human games study such as Caillois' Man, Play and Games[6], study of video games continues through a number of academic journals and industry conferences such as Game Studies[7] and Game Developers Conference[8]. A number of courses and trade programs teach fundamentals of game design and development using a variety of textbooks from industry veterans such as Chris Crawford[9] and Rollings and Adams[10].

Much research is done into game design for entertainment or engagement, narrative structure of games, the community of players which are communicating and collaborating through multi-player shared experience. These subfields are focused on improving games as games, or studying the effect of these games on other aspects of society and life. They are not relevant to our goal of the transfer of interaction methods from games into different applications. Studies on the design of interactions in games are highly relevant, as is research on the different input methods and hardware used to interact with the games.

2.1.1 Game Interaction

Interaction methods used in games are highly applicable as the source for improvements to interfaces are identified from games. Fagerholt[11] studies first-person shooter (FPS) games, detailing the elements on screen which convey information to the player. A survey of FPS games to ascertain the feedback methods used was conducted to catalog these elements. Fagerholt then classifies these interface elements along two axes, determining if they are spatial or non-spatial, and fictional or real.

Elements presented in relation to the three-dimensional space the avatar is moving in are spatial elements, in contrast to elements shown on a 2D plane above the action. Fictional elements are meant to be understood as part of the game world, with the avatar and other characters reacting to them, in contrast to elements which are only visible to the player. An in-game street sign would be fictional and spatial. The HUD showing the

amount of ammunition left is non-spatial and real. Other commonly used interactive elements from FPS games were classified illustrating the different classifications.

The framework extends upon the work by Fagerholt and extends it to associate the specific feedback in relation to an interaction which is driven by the motivation of the player in a specific task. This connection to a task brings a purpose to the element and can therefore subsume the reason an element is presented within a specific category. We do not consider the distinction between fictional and meta non-spatial elements as they are the same for the framework's purposes.

Other research focuses on interaction by studying player actions and how they translate on-screen into actions in the game world. Gregersen and Grodal[12] study the input space of primitive actions, called *P-actions*, and how they are mapped into game actions. They compare the game controller and the keyboard, and record the dominance of a particular "standard camera control scheme" which maps a common set of P-actions in most games to the same actions in their respective game worlds. They go on explore other P-action mapping choices including that different methods of interaction may impact on the immersion of the player in the game. They argue that mapping P-actions correctly can increase player agency. This mapping from P-action to action is vaguely similar to a related part of the interaction model represented later, but does not fully present it in a way that can be described and returned to feedback. The presence of the "standard control scheme" is one argument for using the game interactions in robotics.

Some research comparing how different input methods impacts performance in games has was studied by Klochek and MacKenzie[13]. They developed a number of metrics for identifying the tracking of targets in a first-person perspective, similar to many FPS games. They compared these metrics when the tasks were performed using both a game pad and a mouse to control the view direction. They showed that using the mouse for those tasks was more accurate, and proposed a number of explanations using the metrics to explore the differences in the tasks. The specific results that the mouse was more accurate in tracking may not translate to the robotic tasks. However, the fact that a significant difference was found when changing input methods suggests strongly that different input methods for robotic interfaces will have a similar effect.

2.1.2 HCI and Games

Games exist in an uncommon area when traditional human-computer interaction usability is considered. Nielsen's usability principles[14] of learnability, efficiency, memorability, error prevention, and satisfaction are mostly ignored or explicitly sidestepped in games. Efficiency and satisfaction as defined by Nielsen are placed at odds; If a game played for enjoyment efficiently finished all tasks without effort it would be unsatisfying to the player. Often obstacles which are largely unnecessary are placed in the way of completing tasks which is within sight to lengthen a game. Errors are also not prevented, but created on purpose as game designers specifically seek the effect of failing a task and then later achieving that task to manufacture accomplishment. Some research on games within HCI has focused on evaluating the usability problems which are still apparent within game interfaces, highly relevant to this work.

HCI usability research has also compared different input methods. Kavakli and Thorne[15] compared driving games when controlled by a keyboard, mouse, or joystick and measured user satisfaction and error rates. They concluded that even within the same genre of games, significant differences between games can cause one method to be preferred over another. This difference in both objective and subjective views of interactions based on different interaction methods provides more motivation that changing operator methods for robot interfaces will have a similar effect.

Pinelle et al.[16] studied at the usability of games and developed heuristics from identified problems reported through game reviews on a website. When studying the reviews, they found that usability problems only appeared in any significant number when the games were rated poorly. They also define a set of heuristics for designing good game interfaces. These heuristics are targeted toward the overarching game interface, and not applicable to specific interactions that are extracted here. The evidence that interactions built into the successful games provide a good base for usability reinforces one of the premises motivating use of the framework, and strengthens the choice to study only successful games to discover interactions.

2.2 Human Robot Interaction

The main contributions of this thesis are to the field of human-robot interaction. This field includes all interaction where a robot and a human are involved, including but not limited to preventing human harm from robots, robots as social partners, teaching robots to perform tasks by example, and the expressiveness of robots as interpreted by humans being related to. The development of methods of control for mobile robot or team of robots is the focus here. The scope is limited based on the analogy to game interfaces, where the player controls an avatar in a simulated game world The motivation behind limiting the scope is based on the thesis on games, and the relation to the player controlling the avatar in the "remote location" of the simulated game world.

Many interfaces using these methods have been developed and tested to operate mobile robots. There tend to be two different styles of interfaces which are prevalent: teleoperation interfaces or supervisory control interfaces. Supervisory control interfaces commonly use a map which is either predefined or built as an environment is explored. These types of interfaces are favored for their situational awareness, ability to control multiple robots and varying robot autonomy levels. Teleoperation interfaces, by contrast, focus more directly on a video feed provided by the robot and enable the operator to move the robot and manipulate the world based on the video feedback. Teleoperation methods are more common, as they use less resources, are much simpler in terms of automation, and have reduced sensor requirements of the robot.

Three specific topics in HRI as related to mobile robots are discussed here. Metrics for measuring interfaces, either objectively or subjectively, are reviewed to determine the most accepted methods. Some guidelines that have been already proposed for mobile robot interfaces are differentiated in their design and approach from the framework's approach. Finally a number of specific interfaces for both teleoperation and supervisory interfaces are reviewed to compare the design methods employed in their approach and demonstrate that current interfaces can derive some benefit from game interactions.

It is important that the framework presented here produce interfaces which can be measured to be improvements on the currently used interfaces. A number of metrics for evaluating interfaces have been used previously. Some of these methods will be used in the framework to validate changes brought from gaming interactions are indeed

beneficial. A survey of the metrics used in human-robot interaction by Murphy and Schreckenghost[17] show that there are a large number to be considered. Steinfeld et al.[18] identified key tasks usually performed by robots and developed metrics. The framework uses the common task categories that Steinfeld introduced to generalize the tasks to be improved by using the framework. The framework also benefits from widely used metrics measuring Situational Awareness developed by Endsley[19] and multiple robot Fan Out introduced by Olsen and Wood[20] and further studied by Mitchell et al.[21] Methods of measuring cognitive load like NASA Task Load IndeX[22], and Behavioral Entropy[23] are also included.

2.2.1 Guidelines

This work establishes a method for producing robotic interaction methods by drawing from game interactions. This is in contrast to existing methods which approach design of the interfaces by studying effective current interfaces or metrics and drawing guidelines from them.

Yanco et al.[24] studied four interfaces used in search and rescue robot competitions, using objective metrics such as victims found and total run time and by coding of the activities to a specific set of guidelines developed to evaluate search and rescue operations. After studying these interfaces, they presented a set of six design guidelines. These include providing a map of the robot history, fusing sensor information into a single display to lower cognitive load, using a single window, and allowing control of multiple robots. This represents a bottom-up method of deriving guidelines for interactions to be included in an interface. In contrast, here the method instead uses established well-functioning game interactions to derive new interactions to build a new interface which is then tested for suitability. We also note that many of the guidelines that are proposed by Yanco et al. will be found to already be used in game interactions.

Earlier, Goodrich and Olsen[25] developed guidelines instead in a top-down manner, designing the metrics which need to be optimized such as neglect time, and fan out, and deriving seven principles for efficient interaction which should fit these metrics. Some principles end up being at odds, such as directly manipulating the world and manipulating the robot-world relationship. Again our approach differs from a top-down method because we are drawing from successful game interactions and testing these

interactions using well-known metrics to ensure they are beneficial. We can again note that many of these guidelines can relate well to interfaces in use in video games. When the player is manipulating the avatar-world relationship they are playing a avatar-based game, and conversely they manipulate or request the world be manipulated by units when playing a commander-based game. Along with these they also present attention management through interface feedback and multiple methods for accomplishing the same task, both of which are prevalent in many game interfaces.

A large number of interfaces for mobile robots have been developed over the years. A sampling of them from both supervisory control and teleoperative methods are reviewed here, with special attention made to interfaces which use controls which are inspired from games, or that show evidence that changing only the interface can improve efficiency.

2.2.2 Supervisory Interfaces

A supervisory interface is distinguished by the autonomy level which is provided by the system of robots. The operator takes on a role similar to a supervisor over a subordinate, defining tasks for the robots to perform at a higher level than the teleoperation interfaces discussed later. Normally supervisory interfaces require more autonomy from the controlled assets as the tasks defined require multiple steps, or an effort over time. Many supervisory interfaces can control multiple robots simultaneously due to this higher level of individual robot autonomy. Three of the interfaces which are developed in this work are supervisory interfaces.

One recent novel supervisory interface was developed by Skubic et al.[26] In this research, a interface is presented where the operator both creates a map based on prior knowledge of a remote area, places and controls the positions of robots on that map once created. The robot sensor data is displayed on the map, and objects can be moved around to refine the previously specified map. The interface was validated using an ad-hoc user study at a conference and found that most participants were able to complete the relatively simple labeling and navigation tasks. The interface is interesting due to it's ability to reasonably explain the environment with little prior knowledge, but does not map well to the situation where the world is previously determined as in most game interfaces so the specific method of sketching and refining the environment is not used. Specifically with a significant increase in the number of robots controlled would make it

tedious. Also the environment will be much more complex than that presented. The use of gestures to specify actions is a game-like interaction, but less useful when considering expansion to many more actions.

Another work with a relatable design is presented by Kawamura et al.[27], in which they create a mixed-initiative agent-based architecture. They use a map model and events are detected which trigger alerts prompting operator response in the user interface. The robot sensors is displayed on the screen in a specialized area, and the interface can adapt to diverse sensors and present them in specialized views. They extend an ego-centric approach to sensing which shows the sensed objects in a semi-sphere around the robot in the interface. This interface presents a number of desirable characteristics, specifically the ability to plan events and present multiple tasks on the fly. The interface presented is disjointed and splits the operator's attention, with a surfeit of information in multiple areas. This is an excellent example of an interface which could be improved using the methods developed here.

Heckel et al.[28] developed an interface which is inspired by real-time strategy game interfaces. The RIDE interface allows control of multiple robots and tasking of many robots at once. They source the interaction methods to use in their interface from the different interactions used in the real-time strategy game as possible. This allows them to control a large number of robots within an interface and show feedback from all of them on a supervisory screen. This relates as an early showing that the transfer of these interactions can be useful. Heckel et al. do not go on to compare their interface to an interface without these interactions, however. They also only copy the interactions from a single game, while the framework presented here is a more general method for producing improved interfaces from any number of game interactions.

2.2.3 Teleoperation Interfaces

Teleoperation interfaces are the most prevalent type of interface for mobile robotics in use today, with most of the military and search and rescue robots operating at least partially through teleoperation. In teleoperation, the operator is in direct control of the robot, most commonly with one or more joysticks which is controlling the locomotion of the robot, with a camera feed producing most feedback to the operator. Various interfaces have been developed with different enhancements in recent years.

Keyes et al.[29] present an evolution of a mobile robot interface through many iterations of improvement, performing user evaluations after each step change to identify areas of concern and refine the interface. This interface centers the video in the interface and presents a pseudo-3D version of sensors which shows the distance to various obstacles as well as a map which is being built as the environment is explored. It was found to perform better than other interfaces which presented the same information with less granularity. The study shows that simple changes to the interface for teleoperation can result in significant changes to the performance on a task. The same type of work is developed here.

Nielsen[30] developed another mobile teleoperation interface which uses augmented virtuality to provide a stronger situational awareness than typically available using an interface which only presented a video display. The interface uses a number of sensors to build a map of the environment which is then shown in the interface with a model of the robot being controlled. They showed that using this 3D interface was significantly faster than a more typical 2D interface with the same information available.

Teleoperation interfaces with control inspired from game interactions are also present in HRI research. Kadous et al.[31] developed an interface which adopts the controls similar to first-person-shooter games, using similar keyboard and mouse controls to drive the robot and move a pan-tilt camera. They also use a full-screen display of the video and overlay graphics inspired from video game design. After introducing their interface at a robotics competition, they tested usability by running an experimental user test. They observed that their game interface was easy to learn, especially by those who had identified gaming experience. This is evidence that using interfaces which are similar to game interfaces can reduce training time. Again the interface was built in isolation to other game improvements, and not generalized to a repeatable method as presented here.

2.2.4 Game Engines in HRI

Most modern games use a 3D graphics engine and models to display the information that is being presented to the operator. As it has become more accessible and prevalent, this same technology has been put to use for robotic interfaces. A number of simulators have been implemented using it as well as some other visualization tools.

Simulators commonly use 3D graphics engines in the robotics field. USARSim[32] was developed initially for robotic education and extended later to be used in multiple research approaches. It uses the Unreal Tournament engine, developed by the video game industry, and the low cost and barrier to entry to the physics simulation engine included is considered one of the major advantages. Gazebo[33] is a similar simulator which uses instead the open-source OGRE[34] graphics engine, a graphics-engine which provides no physics. Gazebo pairs this with the rigid body dynamics provided by Open Dynamics Engine[35] for high fidelity simulation. Both of these simulators provide views of the simulated world using a third-person flying camera, but neither are meant for use as robotic interfaces or provide any way to communicate with a middleware for control.

As part of the ROS[36] project, the RViz visualization tool also uses the OGRE engine and can be used to send commands to robots or view data from robots. The main view is a 3D display of the data similar to the world views which are shown in USARSim or Gazebo. RViz is meant for more of a research and visualization tool as opposed to a full-fledged interface, but does provide some controls which can be used for specifying target points or controlling robots.

Through the evidence and direction provided by these works, a number of ad-hoc methods have been used to generate robotic control interfaces. Video games have produced useful contributions to remote control interfaces, with examinations of the different types of interactions and study of player action to triggered actions and look at different input methods. In the robotics field, the study of user interfaces has increased and produced a few game-like interfaces in an ad-hoc manner. Using these studies as evidence and providing early direction, the development of a method for translation of computer game interfaces to robotic interface will meaningfully advance the methods of producing robotic interfaces, by generalizing and leveraging interactions already developed and studies by the games industry.

Chapter 3

Game Interaction to Robotics Framework

Video games, since the earliest days of SPACEWAR[37], have been creating new interfaces for game interaction. One of the earliest joysticks used for computer games was designed for the PDP-11 version of the game.

A number of valuable interaction paradigms and player interactions from games could be beneficial for use in robotics applications. Out of the many thousands of games that have been created, each has a number of response and interaction methods. To find the interactions which are beneficial for robotic interactions, a guide for robotics researchers and interface developers is needed. The framework developed provides a method to identify beneficial interactions using a target task. Following the framework, a robot system can be deployed which is easy to use for users who are familiar with game interfaces and is more efficient at the tasks required than currently used interfaces.

The framework's five main steps, as previously seen in Figure 1.2 are:

1. Decompose the tasks the robot interface performs into common tasks
2. Choose a game interaction suitable for the task required
3. Translate game interaction elements into robotic interface equivalents
4. Develop robot interaction methods using the game interaction elements

5. Evaluate game-enhanced robotic interface to measure change

The framework starts from a robot task that is necessary for the goal of the operator of the interface. The task is decomposed into common tasks in one of five categories – navigation, perception, management, manipulation, or social tasks. Multi-step tasks can including more than one type of basic task, and could benefit from two or more different game interactions. The framework puts focus on a single task at a time.

Once the specific task category is identified, game interactions which are similar should be identified. Texts which are instructional on game design can be used to ind some interactions, but a much larger corpus of interactions within modern popular games is also available to source interactions. Results from an initial study of interactions gathered from recent games are presented in Chapter 4. The resulting matrix of interactions as related to tasks is shown in Figure 4.3. Choose one of the interactions which accomplishes an analogous task in a game, or generalize a set of interactions which are similar in multiple games.

This game interaction is parameterized to identify the key elements: input, feedback, game objects, and processing. The input and feedback are straightforwardly the input the player provides and the signals which are sent back from the game in reaction. Objects and processing are more opaque but can be deciphered based on the game programming or implied from the interaction itself.

Elements of the game interaction are then mapped to the analogous elements of a robotic system capable of the chosen robotic task. The input and feedback can be added to the user interface of the robotic system with little to no modification. The game interaction might need to be distilled to a common elements if it exists in multiple methods, or a specific implementation can be chosen from a list of implementations.

When these interaction mappings are complete, a new robot interface can be created that uses the game-inspired interface. The specific method used for this is dependent on the robot system being programmed, but a generic interface can be created which plugs into common middleware systems

Once the interface is implemented, the new interface should be tested in a set of experiments to determine the magnitude of the effect on performance, if any. Metrics which are in common use for robot performance can be used to evaluate the interface improvements. These metrics are drawn from established human-robot interaction or

robot performance measures, which are chosen to highlight the performance related to a specific task. In addition to these, other methods of evaluation are drawn from human-computer interaction to measure the ease of use of the interface as well as the workload placed on the user.

If the interface is shown to be beneficial, results should be integrated into the common knowledge and the process can be repeated for different tasks in the same interface, combining many different game interactions to create a composite robotic interface suitable for many tasks. This combination of tasks can be guided by the co-incidence of interactions within the same game interface, which would indicate that they do not interfere with each other.

3.1 Decompose into common tasks

Performing a task using a robotic system can be quite complex, and involve many stages. Suppose an operator is given a mission to find survivors, a typical goal in urban search and rescue operations. The operator will perform many basic tasks in the service of that mission. While some games might have very similar analogous situations, it is more beneficial to decompose the mission into a set of common tasks which can be enhanced separately.

The task categories of Steinfeld et al.[18] are used for much the same reasons as they are chosen there: the framework focuses on task-oriented mobile robots that have tasks performed at multiple levels of robot autonomy. They can be applied to both teleoperation and supervisory control interfaces. Each different category is distinct and separate from the others.

Navigation is involved in any task which moves the robot from one location to another. Navigation tasks are omnipresent in mobile robotics. This task has three subtasks which can be thought of as independent but related. Localization is concerned with the current location of the robot in relation to the other parts of the environment, and includes sensing for creating maps and other reference points. The wayfinding subcategory involves path planning and choice of direction – choosing the best method to get to the goal location from the current position given the locomotion capabilities. Movement is the last subcategory, encompassing any task about actually moving the

robot physically.

Perception tasks focus on understanding the state of the remote environment the robot is operating in. Cameras are common tools for perception, identifying objects of interest or surveilling the remote area. Results from sensors or algorithms which detect movement or sound are also included. This task notably does not include the sensing which is required for localizing, which is included in navigation.

Management tasks coordinate multiple robot resources acting either separately or in a group. This task includes the understanding of the differences between capabilities of each robots under the control of the operator. Understanding the current internal resource status, such as battery levels or malfunctions are also considered management tasks.

Any change performed to the environment by the robot is considered a manipulation task. Opening doors, operating elevators, or moving debris out of the way are common examples. Robots without specific manipulators can still often modify their environment, by pushing open a door for example.

Social tasks are introduced when the robots must interact with humans or communicate in a social way. This task is less common in current mobile robot missions, but may become more prevalent in the near future as robot colocation with humans in the environment increases. Interacting well in a remote environment with some level of automation is key to this type of task. For the purposes of the framework, communicating with humans in the remote environment, either through a selective interface or automatically is considered part of this task category.

To decompose a particular mission, choose the categories of tasks used by identifying the different tasks being performed. Using the survivor search as an example, it involves navigation tasks including all three types. Localizing is needed to discover the structure of the remote location, movement of the robot is required to explore that area for survivors, and wayfinding to return to the collection area when complete. Identifying the survivors found and assessing their state is a perception task. Moving obstacles or clearing the environment to continue searching are manipulation tasks.

Each of these tasks should be looked at independently. Games with similar situations may provide similar situations, but it is still more desirable to use the interactions in those games as their benefits will be applicable more broadly. By instead using common

tasks, interactions can be drawn from a wider subset of games that have similar basic tasks. A driving game focused on getting away from pursuers may have similar basic tasks for movement, but no specified end point so provides nothing useful for wayfinding. Once the mission is decomposed into the common tasks, the interface for each of the tasks can be improved on separately.

3.2 Choose and describe game interaction

Next a game interaction must be chosen to transfer to use in the robotic interface. After a task to focus on has been identified, there are three options for selecting a game interaction for inclusion into the interface. Interactions can be discovered through the study of games, learned from description in game design texts, or chosen based on previous research that proposes or has previously showed benefit. When choosing a game interaction to implement, the objects involved in the interaction should be identified. If interactions are to be discovered from study of games, heuristics are proposed to categorize the game interactions into the correct tasks.

To describe fully each game interaction, a clear definition is needed. Any game interaction provides some type of feedback to the player based on a simulated world state. The interaction occurs over a period of time and can include zero or more inputs from the player. In the course of playing a game, players participate in many interactions to accomplish their goals.

A simulated world is defined by a sequence of states $S_0, S_1, \ldots, S_n$. This world simulation can be generated by the simulation engine E_s producing each state from a prior state S_n and elapsed time dt, $S_{n+1} = E_s(S_n, dt)$. Games are distinguished from simulations by interactivity, the ability of the player to affect the world state. The player interacts with the game using an input device with the goal of manipulating the world state to achieve their desired goal state in the simulated world. A game interaction happens in a loop starting and ending at the player, with the player affecting the simulated world through input to the game engine, and receiving feedback about the state of the world through a rendering function. An illustration of this game interaction model is shown in Figure 3.1. A game interaction occurs as one iteration of a physical-virtual control loop with the player providing some input I to the game through an

input device, and the game producing some feedback F to the player communicating the game state.

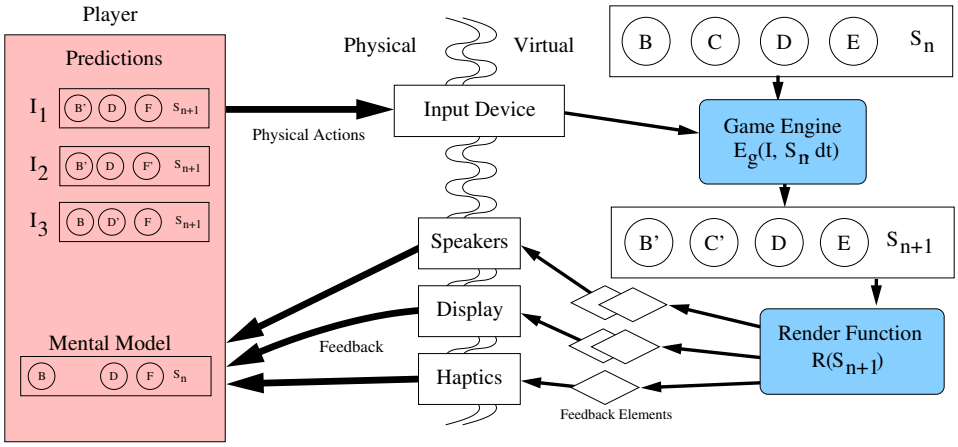


Figure 3.1: Model of a game interaction.

The game engine function E_g produces future game states based on the preprogrammed rules of the game world in response to the current world state and the inputs provided by the player. A game engine E_g is distinguished from a simulation engine by incorporating input I such that it produces the next game state $S_{n+1} = E_g(I, S_n, dt)$. For most modern games, the world state and the game engine are highly complex.

An object-oriented view simplifies the world-state and effects of the game engine. The world-state is a set of m object states $S_n = \langle O_{n,0}, \ldots, O_{n,m} \rangle$. A subset of objects is changed by the game engine based on the input provided. In the case where the player provides no input, $S_{n+1} = E_g(\emptyset, S_n, dt)$ the game engine produces next world states like a simulation. This current goal of the player can be understood as a desirable state of a subset of the world objects G, and the goal is achieved when $G \subset S_n$ at time n.

Once the world state is updated, feedback is provided to the players according to the render function R which produces a set of output feedback events $F = \langle F_0, \ldots, F_k \rangle$ in $F = R(S_{n+1})$. The feedback function translates the game state to a suitable set of output feedback events, which may be presented to the player over a period of time. These events include graphical, haptic, and auditory outputs.

Multiple game interactions can be occurring simultaneously, and the union of all game interaction at each point in time represents all input and feedback which is being

used to play the game. Most interactions are separable from each other, with the inputs affecting specific objects in the world state, and producing specific feedback from those objects. Interactions which share objects or are produced by the same input are considered related.

Some interactions are situational, that is, the engine will not modify the state in response to the input without a specific subset of objects in the game state. This prerequisite state $S_p \subset S_n$ should be specified as well when it is applicable to an interaction.

Using this model, each game interaction can be parameterized as a set of inputs I, which may be empty, a set of involved objects $O = \langle O_0, \ldots, O_m \rangle$, a set of feedback events $F_0, \ldots, F_k$, and the optional prerequisite state S_p. Objects are considered as involved if the game engine manipulates it based on the input, or feedback events included in the interaction are produced based on that object. Note that the objects providing feedback need not be the same objects which the engine modifies.

A parameterized game interaction can treat the game engine and render function as a black box model, with the input producing a change to the world state and producing the feedback provided. It is beneficial to understand what occurs with the modified objects in the world state. Changes to involved objects can be understood as a function $O_{n+1} = f(O_n)$ where O_n represents the understood objects before the interaction, and O_{n+1} the objects as represented by the feedback. This processing should be understood and described, even if not exactly as implemented internally to the game.

One example of processing would be reduction of a player avatar's health due to incoming weapon fire. The player's health object is modified to have a reduced value, and the incoming weapon object is destroyed. In addition to these involved objects, the minimum player health, and any armor being equipped reducing the health change calculation would be secondary objects in the process.

Any method of choosing a game interaction should provide a parameterized interaction definition, and a description of the processing to be able to continue with the interaction to mapping it into the robotic interface.

3.2.1 Interactions from game inspection

Video games as the source of interactions is an easy way to find interactions, but can present challenges to finding a specific interaction. Hundreds to thousands of interactions are performed in a typical gaming session, but most will not be applicable to the robotic task which is being focused on. Some guidelines to help identify the game interactions to focus on based on the robotic task are discussed. Once a game interaction is selected from the suitable found interactions, the parameters of the model should be described. A guide to the common inputs and feedback methods of game interaction elements in modern games is provided.

Task-related game interactions

When considering interactions for use in the framework, an overly large number could be included. Only a subset of these interactions will be useful to translate to improved robotic interaction. Game interactions are not useful in the framework if they cannot be tied to a common mobile robotics task. Using the robot task which was previously selected, the set of interactions can be filtered. Many interactions exist in games and can be related to one of the five categories that have been identified. For each category of robot task, a similar task can be found in games. Finding these similar tasks in games are good guides to discover the related game interactions that can be beneficial for translation to the robotic interface.

When considering the navigation category, a variety of game situations are very similar. Almost every game involves the basic movement of the players' units or avatar throughout the world, providing a rich set of movement interactions to consider. Many games also provide tools to aid localization, including radar screens and maps, geographical or artificial landmarks. Reaching or delivering an item to a goal location is common in most adventure and role-playing games, and a higher level of wayfinding is present in some real-time strategy games as well. In other games, hints are presented in the world to coax the player towards a particular direction or dissuade the player from undesirable areas. This is often for story or technical purposes because an entire world cannot be modeled and created. Any interaction which indicates the "correct" direction or reduces the mental load on the player for wayfinding should be considered.

Perceiving and understanding the environment is a strong component of many video games. A significant percentage of game interactions are involved in showing a window into the game world, and a method for predicting the future state of the world based on the game rules. Key interactions are those which produce more information about the world state than would exist in the real world.A significant percentage of game interactions are involved in showing a window into the game world, and a method for predicting the future state of the world based on the game rules. Interactions that produce a stronger understanding of the state of various objects, like graphical indicators that an object is manipulatable, or shown distances between two objects, are good interaction candidates. Other interactions to be categorized for these tasks include feedback presented as part of some "advanced technology" in the game's narrative such as friend/foe indicators, radar screens, or x-ray vision. Sensor interfaces presented in-game like a stealth indicator or radiation display also fall within this category.

Handling multiple people or units are categorized under management tasks, and are quite common in strategy games. In most strategy games, the player is tasked with tens if not hundreds of units to command and coordinate. Interactions enabling management of troops into groups and coordination between different types of units are good candidates. Interactions which involve the display of different abilities or statistics of a selected unit should be considered. These interactions which allow control and status checking of multiple units at once will eventually increase Fan-Out[20], a key metric for management.

In first-person multi-player games, units can communicate to coordinate actions or direct each other for more effective results and some interactions could be considered as management. Interactions that enable this interaction, like a squad leader view where objectives can be communicated, should be used. As internal status is also considered under management tasks, interactions showing health indicators or status of equipment are also useful sources.

Manipulation of the game world is required in many games, either through simple tasks such as pushing buttons or opening doors, or more advanced placement of objects. In many games, the physics engine supports pushing debris or other world objects as well. Interactions that may be beneficial are usually involved with collecting items from the world, such as picking up rewards or items required for advancement. Interactions

which help identify objects for which affordances for manipulation by the player would otherwise not be obvious will translate well to manipulation type tasks.

Social tasks are a significant part of many game genres, especially in story-driven games. In many of these games the only way to progress the story is to affect a social interaction with another player. These games also must solve the problem that the full range of social interaction is not possible to account for. This means that some of the same type of limited social interaction must be both controlled by the player and be meaningful in the game world - a situation which translates well to the definition of social tasks that robots perform. Interface elements for presenting options for communication are good starting points for interactions here. Journals, quest logs, and other interactions integrating the story so far as well as the immediate goal. Recognition of key non-player characters and the importance of items are also social interactions that can be brought for consideration for social tasks in robotic interfaces.

When searching for interactions from games, sometimes new interactions or similar tasks do not appear. Games which have viewpoints very different from first or third person, or ones with strange control schemes could have no new interactions to be found. Many puzzle, competitive, or educational games for example are unlikely to produce beneficial interactions. Some puzzle tasks can map well to manipulation, navigation or perception tasks, or may provide useful interactions when a higher level of autonomy is desired.

Each game should be examined for possible interactions from multiple angles. As new games are developed and creating new genres and mixing old genres together, as well as innovating with new interaction methods and feedback styles to produce novel and effective game interfaces. Using this competition, experimentation and innovation should provide a steady stream of new interactions to consider for use in robotic interfaces.

Identifying inputs

Input is typically provided to games by using either a standard keyboard and pointer device, or by using a joystick controller. Mass-market gaming consoles have increased the use of the joystick controller. A selection of the controllers used today are depicted in Figure 3.2. These controllers provide a large number of distinct input actions. Most

provide at two or more analog joysticks, paired with 6–12 push buttons. These joystick controllers evolved over time to provide increased flexibility, efficiency and ergonomics of the controllers as various companies competed to produce a flexible and desirable input method for game designers. The number of buttons available has had a generally increasing trend with further iterations signifying an increase in the familiarity with games and game controllers or in the capacity of the player to use them.

When examining the most recent iteration of game controllers, commonalities are apparent. One of the high-precision analog joystick inputs are placed as the primary controller in a position where the thumb naturally sits on them with the standard grip. The number of buttons on the controller on the last last two iterations of the most popular consoles have been the same, with 8 buttons or triggers and 4 more arranged in a directional pad easily accessible when playing. Each controller also has three more buttons that require repositioning of the hands, normally used for functions unrelated to the game (such as the "Xbox" button or "PlayStation" button which exits the current game).

(a) Playstation 3 (b) Gamecube (c) Wii (d) Xbox 360

Figure 3.2: Controllers from various recent game consoles.

In addition to joystick and keyboard inputs, two new styles of input have emerged in the recent years: body control and motion control. In body control the entire body of the player is used as an input to the game, using computer vision algorithms combined with new distance sensors such as the Kinect[38]. In these games, the arms, feet and torso movement and positioning is used as an input. The second iteration of these sensors can distinguish players from each other as well as accept gesture input and even measure heart rate[39]. Motion control has also emerged in recent years, partially in response to sensors in smartphones and notable use as a method of input for the popular Wii game system. In motion control input, an accelerometer and/or gyroscope

contained in the controller is used to produce up to 9 axis of input for the game to use based on the rotation and position of the controller.

As smart phones and larger high precision touch surfaces become more prevalent, touch control or direct manipulation input is in increasing use in games, especially with mobile and tablet games. Touch input can be thought of as similar to mouse input, with multi-touch surfaces allowing multiple actions, and multi-finger gestures also being available.

Feedback Types

Feedback is the most essential part of the game interaction, as it communicates the world state to the player. Three different modes represent the majority of feedback: graphical, aural, and haptic. Each can be combined with others or used alone.

Different feedback elements can often be difficult to isolate as related to a specific interaction because it is all combined on output. When determining feedback included in a game interaction, consider the effects which are presented to the user and what game state they are representing. Feedback elements derived from involved objects should normally be included in a interaction definition.

The primary mode of feedback for video games is graphical. Practically every video game uses graphical elements to guide the player through the game. This partiality can even cause problems for game players with impaired vision, although specific games and methods have been developed to help[40]. Graphical feedback can be classified into many different categories.

Earlier games were likely to use paragraphs of text to communicate the state. Later, low-resolution pictures of scenes were displayed and could be explored. Two-dimensional sprite based games represented a significant improvement, with relative positioning and the modeling of the trajectory of objects in the world. A subset of modern games continues to use this two-dimensional model. Three dimensional graphics engines are now the dominant form of producing graphical feedback, rendering entities within a three dimensional world which could be viewed from any virtual pose.

In three-dimensional graphical feedback, there are three major types of entities to consider, guided by the categories from Fagerholt[11]: Diegetic, Geometric, and Overlay. Diegetic elements are elements which are rendered as part of the world and represent

physical objects that are being physically simulated, such as the player's avatar, the landscape, and other objects such as crates or barrels. Geometric elements are rendered as a three-dimensional object, but are not affected by simulated physics. Many graphical effects are geometric, used to draw attention to a particular diegetic entity. Overlay elements are rendered onto the screen two-dimensionally, such as health bars or ammunition counts.

It is best to discern graphical feedback based on the objects being represented; ammunition count for a weapon and the health remaining indicator may share a similar look but should be separate feedbacks as they represent different objects. Graphical feedback can often be transitory as well, disappearing with time or only appearing when a particular state is achieved.

Auditory feedback is another area where games provide a significant amount of feedback. Early games only provided simple beeps or monotone blips, but the audio presented by games today has advanced to exists in an auditory envelope in three dimensions around the player. Often integrated into the graphics engine, each audio feedback can provide directionality as well as magnitude and type. When categorizing audio feedback, the feedback can be specified using the sound presented, 3D direction with respect to the player, and volume. As an example, a exploding grenade would play the grenade explosion sound effect, in the direction which can be drawn as a line between the player's avatar object and the grenade blast location, with volume as a function of the distance away from the player. Using this single audio feedback, a skilled player can determine both direction and range. Audio feedback which is omnidirectional like narration or music can be included with no direction needed.

In comparison to advanced haptic feedback used in scientific research, the haptic methods of feedback used in games is coarse. Since the introduction of the Rumble Pak for the Nintendo 64, game controllers often include a vibration feedback produced using an off-axis weighted motor or linear oscillator. When actuated, it produces a "buzz" sensation for the player holding the controller. Most modern controllers can provide varying levels of intensity as feedback. More advanced upcoming joysticks such as the Steam Controller[41] use weighted electro-magnets to provide an even wider range of haptic feedback. When identifying haptic feedback, a "rumble" often is presented based on an event happening in the game world, such as damage to a ship or the player's

avatar.

3.2.2 Interactions from game texts

While any game can be examined for interactions, some work has already been done in the study of games that would be useful to leverage. Many game design texts have generic descriptions of various game elements which can be studied. All that is needed is to translate them into the parameters of the framework.

Rollings and Adams[10] contains a set of guidelines for interactions, and descriptions of some interactions for each genre of game. Other game instructional texts have similar descriptions of a variety of game interactions with study of the successes and failures of each. Other researchers such as the previously referenced Fagerholt[11] have also studied a set of games and categorized the feedback which is presented to the player.

3.2.3 Interactions from previous research

As designed, the game interactions to robotics framework is meant to produce reusable results. First by generalizing the task into one of the common five categories (and three subcategories of navigation), the specifics of the exact task being done are removed. Next the framework uses these tasks and studies game interactions to find similar tasks which are then connected to the common robotic task and described in detail. After these two choices are made the interaction is made more specific to the exact robotic system being implemented and the interface being modified.

Once the first two steps of the framework are completed, the result can be shared among researchers to continue with the framework in the third step. Once the games are studied for the interactions with the common tasks, a matrix can be provided relating the interactions to each type of task, for use in future iterations of the interface. One such matrix is provided in Chapter 4 in Table 4.3.

Further feedback from the rest of the implementation of the framework can be used to inform decisions made for future iterations of the interface, by integrating the results from the interface evaluations performed in the last step of the framework to update the matrix or add annotations relating to the specific use of interactions with respect to specific or common robotic tasks.

Using a matrix or influence from previous research using the same robotic task and game interactions, the study of the game can be short-circuited and the same game interaction research can be used to design new interface elements for a variety of different robotic platforms.

3.3 Translate game interaction into robot interface

Once a desirable game interaction is selected and parameterized, elements from the game which implement that game interaction should be detailed to outline the mechanics of the interaction. Identify the different types of game world objects involved in the interaction: the primary objects and the effects on secondary game objects which are manipulated in the course of the game interaction, and the process to modify those objects to their new state. Once identified, a mapping for the input, feedback, and these objects to real world interface equivalents must be created. In some cases, this mapping cannot be created, which suggests the interaction may not be suitable for real world integration.

To translate a game interaction into a robot operator interface, the game interaction must be understood completely after parameterization. The elements that must be understood for translation are the modified objects, secondary objects, the processing taken to modify the objects, and the prerequisites for the interaction. Involved objects are the primary objects in the interaction - either the object whose change initiated the interaction if it has no inputs, or the objects which are updated, destroyed, or created as part of the interaction. Secondary objects perform some purpose in the interaction, but are not modified; They could be objects which contribute to a feedback or change the nature of the modified object. All of these objects exist in the game world.

Once all of the objects, prerequisites, feedback, inputs, and processes are understood, the next step is to map interaction elements to the real world. A limited set of real-world objects and possible stored information exist to use for the mapped interaction. Some objects also might be left untranslated, and exist only virtually. For each primary object in the interaction, one of three mappings should be chosen:

- A real-world object identified as a stand-in for that object. One real-world object might represent more than one objects from the interaction. If an object is chosen

as part of the interaction, a non-specific real-world object such as "a target robot" can be specified if it is possible to produce the needed information by choosing in the robot interface.

- An algorithm for producing the object or object data from sensor data available to the robotic system configuration is chosen. A health status object may be represented by a function of the output of a system status algorithm, for example.

- The object is ignored if there is no suitable real-world object or the information can not be produced. Objects which violate laws of physics, such as a gravity manipulation weapon, are easy examples. Another example might be a count of remaining targets when the number of targets is uncertain as in many search and rescue tasks.

Next map the processing in the interaction to to robot commands, algorithms, or real-world physics. Processing which cannot be mapped can be discarded if it is still possible to produce desirable results. For example, if the processing would move the object which is mapped in the object mapping to the robot, it would be mapped to robot commands for moving the motors. Objects which would be affected physics simulation in the game engine can of course simply be affected by real-world physics. Processing which prevents the avatar from colliding with a wall would be replaced with object avoidance algorithms, or could be discarded if the interaction is not affected by the avoidance.

When mapping these objects and processing, it is desirable to preserve as much of the original interaction as possible, to produce a similar type of interaction when interacting with the robotic system as the in the game. Objects which are "controlled" by the player should be mapped to controlled robots. In some cases, the algorithms which are used in the game can be transferred directly to the real-world. One specific example are path-planning algorithms which do not exploit *a priori* knowledge of the environment.

The inputs and the feedback are excellent candidates for direct translation into the robotic system with little to no modification. In a majority of cases, the same input devices can be used in conjunction with the current robot control system or an auxiliary system which produces commands intended for the current system. Feedback can be

translated with almost no modification as well, with graphical overlays and widgets representing the objects being included on the screen being used for the normal robot feedback.

Keeping these elements from the game interaction the same when translating them into the robotic interface has a number of advantages. Changing the input or feedback drastically reduces the similarity of the interface to users. Any familiarity with the interaction that could be transferred to the robotic interface if the interaction is known to the robot operator would be lost to a degree if these external interface elements are changed. The player experience is also one of the areas that was developed and refined when the game was developed, often with significant funding and research. If they cannot be mapped directly to a robotic interface, attempting to preserve them as intact as possible is preferable for these reasons.

It may be the case that an object which is crucial to the interaction, or a process which is essential for the interaction or feedback to be meaningful cannot be mapped correctly. The algorithm or sensors which are required to produce the effects are not available or nonexistent. Game engines are areas where advanced AI is developed, which often requires information from the game world which is known to the engine, but cannot be known when the robot performs a similar task. In some cases, additional preconditions might be placed on the interaction to continue, such as *a priori* knowledge of the environment. In other cases, this can spell the end of the translation of this interaction, as there is little remaining of use. The amount of feedback which is still possible to use combined with the number of robot commands which are being produced are one measure. If there are few of both, it indicates that the interaction could have little relevance to the real-world.

When this stage is complete, the game interaction has been specified completely, and the items which are required for the robotic interaction to occur in a similar manner has been defined. These are all of the items required to implement the common robotic task which is was described in the first step by using the game interaction which was chosen using the second step of the framework.

3.4 Develop robot interface

Once interactions are mapped into a robot system, the interaction method can be developed using these mappings. This proceeds in a typical way that the robotic interfaces are built and is dependent on both the specific mappings to algorithms and hardware that were selected in the previous step, and the robot system architecture the interface is being built for. Some guidelines can be followed to make the evaluation of the implemented interaction easier. A suitable base interface for the robot or robots being controlled should be selected or developed.

In the next step, a comparative study of the interface with and without the new implemented interaction is suggested. An existing interface to accomplish the same common robot task that is implemented in the interaction is used as the existing comparison. If it is not possible to accomplish the task that the new interface will support, the testing methods available will be limited. Often an existing interface for performing the task already exists and can be used for this purpose.

An interface can be evaluated as suitable for use as a base interface based on a few different factors. The interface should be usable to complete the task which the new interaction is being developed to complete. If the functionality is not available in the base interface, one option is to add the functionality in a way similar to the rest of the interface. The base interface should be of similar quality to the new developed interface, as the quality can affect the perception of usability of an interface.

Using a simulation system for the robot system is highly beneficial both when developing the interface and for testing of the interface. Using a simulator can provide an easy to setup and repeatable world for the comparison of the different interfaces, ensuring a fair comparison of the interface without inherent differences in the test runs. A simulator is also portable, and can be tested using remote structured testing as described in Chapter 5 gaining the benefits and expanded possible participant pool.

When implementing improvements, many interactions benefit when combined into a single interface. To identify interaction pairs which are coordinated, the playing of the game they are extracted from can be examined to guide decisions. Game interactions which are often used together are candidates for including together in a single improved interface to be tested. To quantify this temporal locality of use, observe the game being

played normally and note start and end times of different interactions. When interactions are often used by the player in sequence, or triggered together, they are considered good candidates for bundling. The purpose of each interaction in relation to the robotic task should be considered as well as the viability for translation to robotic interaction. Similarly, if some improvements can not be translated to the robotic interface due to unavailable or impossible information, a failure to map objects to real-world equivalents, or an unfeasible input method, it may be possible to segment the interaction into component interactions. Some of these interactions can then be implemented.

3.5 Evaluate developed robot interface

After the robotic interaction is implemented, it must be evaluated to show that it has affected a change in the control of the robotic system. Ideally the changes which have been developed will also show significant improvement. These tests are used to show that the inclusion of gaming interactions provides a significant effect for the operator of the mobile robot or team. Some of the methods which can be used to accomplish this task are detailed here, providing a toolbox of methods for testing the interactions. These methods are related to the game interactions specifically for guidance.

A variety of methods have been proposed for evaluating human-robot interaction. Murphy and Schreckenghost[17] note 42 metrics which have been used in recent years to measure some human-robot interaction. Only a subset of these interactions will be useful for evaluating robot performance in each of the different common task areas. Guidance is given on metrics to use for different of task categories, and advice on methods of evaluation which apply to all interface changes regardless of category.

Evaluation should be carried out in two steps, starting from experimental trials to ensure a workable interface. Next usability studies of various sizes and rigorousness provide significant evidence of improvement on specific tasks related to the initial task desired for improvement.

3.5.1 Metrics

A key component of evaluating a user interface is choosing a set of metrics to test the interface with. As identified by Steinfeld et al.[18], two categories of metrics exist: common metrics applicable to all robotic interfaces, and metrics targeted or specific to the task being completed. Having an interface that is developed based on one of the common tasks provides strong guidance for the task-based metrics to use when evaluating. The set of common metrics can also be filtered based on the type of improvement that is expected from understanding the game interaction.

Once a new interface has been created for completing a task, it should be evaluated based on its efficiency and effectiveness. Efficiency of a robot interface is defined here as a natural adaptation of efficiency of a user interface from human-computer interaction. Typically a more efficient interface performs the same task in a smaller amount of time. More efficient interfaces use less of an operator's time, so the operator can complete more tasks or interact with more robots in a team. Effectiveness in contrast measures how well a task is completed – interface changes often make it possible to complete a task more completely or with better results.

Effectiveness and efficiency can confound each other, with a less effective interface producing results quicker. Each of the metrics will be impacted by the effectiveness of the participant during a test, so the level of task completion might require consideration. While the two metrics can be considered separately, the efficiency measures are affected by reduced effectiveness favoring ineffective behavior. Total task time is lower if an exploration task skips an entire section of the area, for example. The efficiency metrics can be handicapped by measured ineffectiveness to compensate. In the exploration example, a suitable amount of time could be added for the missed section.

Common metrics

Most metrics shared across all tasks are focused on efficiency, as it is easy to measure resources or time taken given any task. User interfaces that are considered here, when paired with the typical tasks of a mobile robot are very open ended. Multiple possible paths could be used to complete the same goal and would be equally effective. Three different recommended metrics are presented here measuring efficiency of the mobile

robot interfaces, with increasing levels of complexity.

The first is *total task time to completion*. This metric is popular and produces easily comparable results. It is best when considering tasks that contain little to no variability of the correct choices for robot control. When tasks are materially the same, difference in time to complete the task can more easily be attributed to the interface. Non-repeatable tasks like exploration of unknown areas should be tailored to have similar difficulty, and an average over a representative sample of task measurements should be used to remove the confounding factor of minute task differences. Completion time is the preferred and most direct metric; it is results-oriented and additionally often the simplest metric to measure. One good example of experiments where the total task time to completion is an effective tool is re-tasking of a UAV, where the interface used can make this a complex or simple task. Missions where the total task length is long but the interaction of the operator is short or the task length is out of the operator's control, such as responding to external stimuli, can produce inconsistent results when using this metric. In these scenarios, it cannot be expected for the interface to lower the external factors in the delay time.

Cases where the total time is long compared to the time interacting with the robot system can use *task interactivity time*. Two states must be identified and separated in the operator's interaction with the interface. In the **interactive** state, an operator is providing input or receiving feedback from an interaction to modify the actions of the robot. Coming from the game interaction, the user is manipulating the input device to affect change or interpreting feedback on the game world objects to determine their next action. In contrast, the **monitoring** state occurs when the operator is waiting for the robot to compete work already specified, whether alert or performing another task. Using these two states, the task interactivity time can be measured as the sum of all time in which the operator is in the interactive state for completing the task. Using this definition, task interactivity time is strictly less than or equal to total task time to completion. Using this metric, the focus can be placed on either a whole robot team, or each robotic asset could be considered separately. Having some monitoring time in the interface is desirable, especially in multiple robot systems, as ancillary tasks can be completed, or other tasks can be completed in parallel. Monitoring the robot's progression without input can also increase detection of issues with the task completion

as coordination between control and perception is less required. One example of a task this metric is preferred for would be a package or material handling task where the targets arrive at unpredictable times. The total task interactivity time metric would show, the more efficient interface lowering the task time, when the total task time may not decrease as work does not arrive based on the efficiency in completing the task.

The motivation to control more robots simultaneously influences the final efficiency metric. The *number of interactive frames* counts the number of times the interactive state is entered during a task. The task interactivity time of an operator could be very small, but spread out over a large number of interactions. A steady stream of very small interactions is often distracting and can prevent the operator from performing other tasks that would be otherwise useful to engage with. The operator can either switch between tasks, which has a cognitive cost, or simply wait for the repeating task to become interactive again. Figure 3.3, illustrates two interactivity profiles which are equivalent in total interactivity time. A more ideal interaction would combine interactions allowing for significant neglect time where another task could be switched to and effectively handled, similar to the second shown interaction profile. A count of interface interactivity periods an operator takes separates the "small adjustments" type of interface from an interface which is more focused on batching changes.

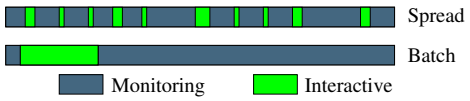


Figure 3.3: Two profiles of activity with the same task interactivity time.

An example of an interface with undesirable total number of interactive frames is typical highway driving. Unless wheel alignment is perfect, the motor vehicle drifts slowly to one side of the road, and requires corrective action at regular intervals. Each of the intervals themselves is small and contains downtime in between. The frequency of these interactions means that driving is essentially taking up all of the available cognitive power of the driver by leaving only small sections of idle time in which to complete other tasks. This type of interaction has small total interactivity time, but also has a large total number of interactive frames.

After effectiveness and efficiency of an interface, ease of use should be considered as an additional metric. Time to learn a new interface is a major consideration of ease

of use. One way to measure the speed of learning an interface is to track unintended consequences over time when the user is presented with a new interface. Unintended consequences occur in games when the world model the player has formed is incomplete or incorrect, and the actions which are performed in the game world differ from the prediction as a result. Typically the player will reverse the decision if possible, or have suffered negative results. An ideal expert user will cause no unintended consequences, predicting correctly the actions performed by each interaction with the interface. Novice or inexperienced users perform more unintended actions, and must be reversed, refined, or canceled. As the user becomes more aware of the actions resulting in the interaction, they should produce less of these unintended consequences. Unintended consequences can be measured by detecting action cancels or reroutes, detailed analysis of the behavior of a user as they interact with the interface, or by self-reporting by the operator. If the number of unintended consequences is tallied over consecutive time periods, downward trend should appear, and the time to achieve a level of familiarity with the interface can be inferred.

One method used to artificially lower the learning curve in games is the training level or area. This area in a game interface is an area where actions are available with little to no consequences to progress, to present the player with more time to become proficient at executing specific interactions. For robot interface evaluation, a similar tutorial area can be created. Simulations can make this easy and repeatable until a new operator feels comfortable with the interface elements, and measuring the time spent in this area gives a measure of the difficulty of learning the interface. These tutorial levels also provide a very useful role of introducing the interface elements if the study is performed in a self-guided manner as suggested by the remote study framework in Chapter 5.

Ease of use is often not measurable by examining directly the data provided by the operator and robot, or can be overly complex. In these situations, employing subjective measures to discover and compare interfaces, with surveys and questionnaires administered after the fact can be the best option available. Two different types of these are effective in measuring user experience: subjective questionnaires and task load surveys. Subjective questionnaires are task-tailored questions which can be performed during or after the task is completed. Task load surveys are similar but are designed specifically

to measure the difficulty of a task under multiple conditions. The training and novelty effects are hard to eliminate in these surveys when presenting one interface after another, so certain methods can be used to eliminate them.

The most direct way to judge a operator's experience can be to ask their opinion. Questionnaires are a valuable tool for gathering this type of feedback. Questions can ask directly about differences in the interactions which were performed, asking for preferences between the methods experienced, or judging each experience independently on similar scaled questions and comparing the scores. Ease of use can often be judged indirectly by asking questions about the operator's belief in their performance. Operators who are more challenged will typically believe that they performed worse than they actually did, and operators who are not challenged inversely believe they are performing well. Questions can be designed to specifically target the portions of the interface that are of interest to the researcher. If desired, developed questionnaires such as the Questionnaire for User Interaction Satisfaction[42] can be used or referenced for guidance.

While looking similar to user questionnaires, work load surveys are both more focused on results and targeting a specific metric. Work load in this case is defined as the cost of accomplishing the task which is presented to the operator, in multiple different dimensions. Work load surveys are designed to measure these dimensions of effort expended by using a survey administered either during or after the task is complete. The NASA TLX[22] metric has been tested as one solution in multiple scenarios to gauge the task load which is subjectively experienced by the user of the interface. The NASA TLX consists of a number of questions which separately query each dimension, and later compares dimensions against each other to determine which elements contributed the most to the overall work load assessment. It also has the advantage of a relatively short length.

The NASA TLX is not without downsides. Training effects should be eliminated, as they artificially increase the perceived work load. Results from multiple studies to suggest that the metric cannot be used to compare between multiple users, as the perceived load is highly subject-specific. A within-subjects design is therefore recommended to compare interfaces by judging a difference using the same user. With multiple tasks by the same subject, this type of design introduces confounding factors that must be considered. The context effect can cause a second TLX survey to be judged differently

because the operator has knowledge of the survey. This combines with novelty effect, in which the new and different method is preferred simply because of contrast to the currently known method. One method of minimizing novelty and context effects is to administer surveys within the experiment, before context effect has occurred and before a second treatment is performed introducing the novel elements. Another minimizing method is using a within-subjects protocol and randomizing the order of the treatments to average the benefit and detriment of these confounding factors.

Another method for measuring work load is Behavioral Entropy[23]. While the NASA TLX and other work load surveys can be subject to psychological biases, Behavioral Entropy measures more directly by using a model of the operator. Given a predictive model of the operator's desired actions derived from their actions completing a similar task, it uses the deviation from the interaction which should be produced by that model and the actual interaction the operator executes. The error in prediction is then used in a histogram to generate a nonparametric estimate of the model's prediction error density function. Then the information within this function is measured by extracting the entropy and is used as a metric, with the minimal workload produced when the model predicts the actions of the operator exactly. This measure also has the advantage of working in real-time, giving a view of the workload which can be used to identify specific interface elements causing issues.

Suggested task-related metrics

Given that an existing robot system, the first tasks to consider evaluating a new interface on should be the original tasks that were decomposed in the first step of the framework. If there are measurable results from these tasks, consider the effectiveness and efficiency of the new interface in completing them. Measure effectiveness based on how much of the task can be completed or what percentage of tasks can be completed if there is some variation in the task. Compared to the previously discussed generic metrics, task-related metrics are more likely to include effectiveness measures. Using the common task metrics can be beneficial if the original task is difficult to replicate, or more generalizable results are desired.

When looking at a localization problem, the effectiveness of the algorithm is usually dominant over the interface, but the communication of the location to the operator

or the understanding of the inaccuracy of the global or local location can be affected by interface changes. For most navigation tasks, metrics including successfully avoided obstacles, deviation from a planned route, and the percent routes completed can be used as effectiveness metrics. Using time to completion also works well for navigation, with similar robot capabilities. Effort in wayfinding can be measured by noting the time spent traversing areas which are not on the path to the goal point. Time spent avoiding or overcoming obstacles can be used to judge movement effectiveness in a teleoperation context, or total time spent specifying movement tasks in a supervisory interface.

Perception tasks are varied, and relate to the sensors which are available on any particular robot platform. They can be split into tasks where sensor data is interpreted, and those where more information is obtained actively. Identifying targets in either case can be judged based on the number of targets detected out of a total number of possible targets detected. Search while the platform is stationary is also included in perception tasks, where the effectiveness metric is measured in the coverage as a percentage of total possible coverage, or efficiency as percentage of new information retrieved.

Situational Awareness (SA) is a group of perception qualities which are desirable for mobile robots that has been examined for use in both teleoperation[43] and supervisory control[44], with a higher level of situational awareness being correlated with accurate or efficient robot control. One commonly used technique to measure SA is the Situational Awareness Global Assessment Technique[19] (SAGAT). Using this technique, the task is interrupted and the operator is asked questions about the situation that they were in at that point in time. After the task is complete, the answers are evaluated based on what was actually happening. This introduces a large effect in the interface which may be undesirable, so modifications of SAGAT exist where the task is not halted, and the questions are asked as a secondary task.

Fan-Out as defined by Goodrich and Olsen[20] is one of the key metrics of when considering management of multiple robots. Fan-Out measures how many robots can be effectively controlled by a human simultaneously, with higher numbers preferred. Average active robots is a related metric which is many times easier to measure. An alternate metric which is more interface-based is intervention response time, which is the delay in retasking a robot after it needs robot intervention. Intervention response time can optionally include the time to notice a robot needs intervention. As management

also includes internal state of robots, one metric for an interface is time to identify a robot by features or status, which can be obtained using a quiz method and measuring the time taken.

When manipulating the environment, mental task load increases as the operator computes the position of two objects and the effects of movement of both. In manipulation tasks, the earlier work load surveys should be considered strongly. Effectiveness can be measured by percentage of manipulation tasks completed successfully, or alternately the amount of contact errors can be counted. Contact errors occur when a robot or held object unintentionally collides with an object or the environment. An interface that reduces contact errors would be considered more effective.

Metrics which evaluate Social interaction of robots are less common, and in the interactions which are human-controlled, the common efficiency metrics are likely best. Effectiveness can still be measured based on the number of interactions which are performed, and the availability of the correct social interaction while completing a specific cooperative task. Measuring the number of uses of social tools can also indicate the effectiveness, as well as considering how many times the social interaction was determined to end with a successful result.

Using some combination of these metrics and the common efficiency and ease of use metrics, the impact of the modified game-inspired interaction can be compared against the base interface chosen. The user interface elements can effect the work load, ease of use, training time, situational awareness, and/or user satisfaction along with the efficiency metrics of time to completion, total task interactivity time, and total number of interaction frames. These usability study metrics are a non-exhaustive set of valuable tools used to show the results of integrating the game interactions into the robotic interface. The methods for measuring these metrics can be as simple as a trial run by an experienced researcher to a multiple-month user study.

3.5.2 Experimental Trials

Initially an evaluator need only consider the viability of a user interface to accomplish the task it is designed to perform. When testing a new interface, this first question of whether it can be used for the task must be answered. Often this is determined by the capabilities of the robot that has been selected for the mission. If the correct sensors and

manipulators are not possible to activate through the interface, the interface may not be able to complete the task. One example would be if a particular feedback obscures vision, a perception task may become impossible to complete.

An analogous type of testing occurs in game development. Games are often play tested for hundreds of hours before they are delivered to ensure that the game engine rules and interactions do not prevent a player reaching the end of the designed narrative. At the same time, measures are made within the game testing to determine problem areas within the game narrative, where players have a harder or easier time than intended, or become confused.

Testing that the robot system with the new interface in an experimental trial can be completed on the initial task to provide similar feedback. Sometimes it is impossible to duplicate the intended task, due to danger to humans or hardware. In this case, reasonable simulations or approximations of environments and tasks can be substituted such as the USAR test arenas[45] for search and rescue robotics. In addition to simple boolean viability, some metrics can be used during the experimental trial. These metrics can be compared against the same metrics for a base interface, if available. For efficiency metrics, a reasonable base number for comparison can be obtained, and can validate that expert users of the final interface show improvement. Of course the interfaces can be measured using objective metrics such as number of actions required to complete a given task as well.

In some instances due to time and resources, experimental trials of the interface may be the only tests which are completed. If an interface enables a new interaction, the benefits of the new interface are proven with successful trials. Alternately, these early experimental trials are a good early indicator if the interaction is not beneficial, as expert users should show an improvement. This is preferred because experimental trials are simpler to set up and less costly than usability testing.

3.5.3 Usability Experiments

After the interface is vetted to complete the action and the researchers are happy with the improvements, more rigorous testing in the form of one or more usability experiments should be done. Usability studies have been a popular method of testing since the 1980s[46], and are used commonly for human-robot interaction[47]. Usability studies

can scale from a small "hallway test" with few people, to a full study approved and administered by multiple researchers. In the usability study, the operator's interface use is observed performing a task, measured according to the metrics chosen for the task.

Compared to typical user studies, a typical human-robot interface study will usually employ one or more metrics, and vary the interface between the subjects to determine the difference caused by the change in interface. Decisions made about whether to have a single participant test both interfaces, using simulation or real robots, and instrumentation of the interfaces can affect the types of tests available.

Variation across participants can be very large with robotic interfaces due to different amounts of experience with robots, natural ability with remote systems, and other factors. Using game interactions can increase the variability by adding game experience as a factor. Using a within-subject study is desired to remove the variability between participants if possible. Regardless, the game experience of each participant should be discerned and correlation of results to experience should be tested.

Using simulation is beneficial to researchers as it provides for a simpler study setup, and repeatable and predictable results for sensors and manipulators. Simulators can not reproduce the world accurately and unexpected results can appear in less-controlled situations that could invalidate the lab results if not careful. In game interaction inspired interfaces it is particularly apt as many simulators are built from the same game engines used building games. In the case of robot interfaces, simulation convenience should be weighed against the fidelity of the simulation causing issues when an interface is transferred to the real world.

If possible, the interface should be instrumented for a user experiment, making it possible to record all input, data and feedback for later examination and analysis. Instrumentation built into some middleware systems can facilitate replaying data and input, showing the state of the system at any time and rewatching interactions as they occurred. This type of recording is valuable but loses a lot of value when the base interface cannot be instrumented, which is common for off-the-shelf robot interfaces used as a base comparison interface. If an interface cannot be instrumented, a usability lab can be used to record the participant and their inputs and feedback.

Usability studies can vary in complexity from "quick and dirty" to massive, expensive

affairs. In the most simple study, a single person could be asked to do a simple test of the system on a whim. Smaller studies of tens of subjects are common in human-robot interaction, a short survey of studies in the last decade averages 20 participants. In small studies like these, statistical results are possible with a large change, but smaller changes are harder to detect. Larger studies are common with simulation, which averaged 10 more participants than their real-world counterparts. Sizes of studies must be balanced against effort and cost considerations, but larger studies are preferable.

If a large participant pool is desired, simulation is used, and instrumentation can be included in both interfaces, a remote structured robot study framework such as the one as presented in Chapter 5 can be used. Using a remote usability study has the advantages of a larger possible participant pool, automated administration of the testing, and ease for administrators. The costs usually associated with remote studies are greater setup and debugging time before starting the study and less control of the environment where the study takes place.

Once the study is completed, analysis of the data through the applicable metrics should produce results showing whether the interface additions created by the game interaction are beneficial. The resulting interface can be used in the field to control the robot system. Statistically proven interactions for specific tasks should be contributed back to the community for use in creating other robot interfaces. If the robot system identified more than one relevant common task when decomposing robot tasks in the first step, the interface can be iterated through the framework for additional tasks or additional game interactions for the same task, which shows viability of combinations of interactions. Given the results from multiple studies, a set of general interface interactions and combinations of compatible interactions for common robot tasks is created which increases the pace of development for usable robot interfaces.

3.6 Navigation task example

To illustrate using the framework, a full example to connect the entire framework together and demonstrate how the steps connect to each other. The robot system is a four-wheeled skid-steer robot that is configured with a grip manipulator on the front. In this scenario, the robot has retrieved an item with the gripper in the mission area,

and the operator needs to return with the object to the entrance.

When decomposing the task into common tasks, a navigation component is identified, specifically a wayfinding task: the robot operator wishes to navigate from where the robot is to a specific goal point. To choose a suitable interaction, the matrix in Table 4.3 is consulted and the two tasks identified for wayfinding are considered. The **goal point indicator** is chosen. This interaction has already been described in the game interaction survey in Chapter 4, which can be reused.

Moving on to the mapping stage, the interaction has no inputs. The processing involves two objects: the location of the avatar, and the location of the goal point. The avatar is mapped to the robot, and the goal point mapped to the entry point of the mission. The involved processing can be directly implemented if the robot system provides semi-accurate localization for the two objects. A simplifying assumption that the robot system does provide suitable localization for the robot, and the entry point is easily expressed within the localization's frame of reference. The feedback presented is a geometric object to be rendered within the three-dimensional view presented to the user, and shown elsewhere on the screen when the view does not contain the point. This feedback can be translated back directly because the pose of the driving camera is known. The feedback will be rendered on the screen using 3D graphics calculations.

Armed with the mapping from game interaction to robotic interface, the new interaction can be developed by enhancing the current teleoperation interface of our robot system. A mockup screen is shown in Figure 3.4. A base interface does exist for this task, as the only task requirement is basic locomotion of the robot. Another base interface incorporating some information used in the new interface could be developed to provide an equal amount of information to the operator. This base interface could display the coordinates of the goal location, in one section of the interface.

Once the interface is developed, evaluation is the final step. From the list of task-based metrics which are suitable for navigation, the time spent traversing paths not on a valid path to the exit is chosen for an effort metric. From the common efficiency metrics, the total task time is perfectly suitable to this task, and obstacle collisions will also be recorded. Subjective metrics will also be used, measuring workload using a survey tool (NASA-TLX) and freeform survey questions. To simulate the navigation task, a maze with one exit is setup, and the robot is placed at a one of two points

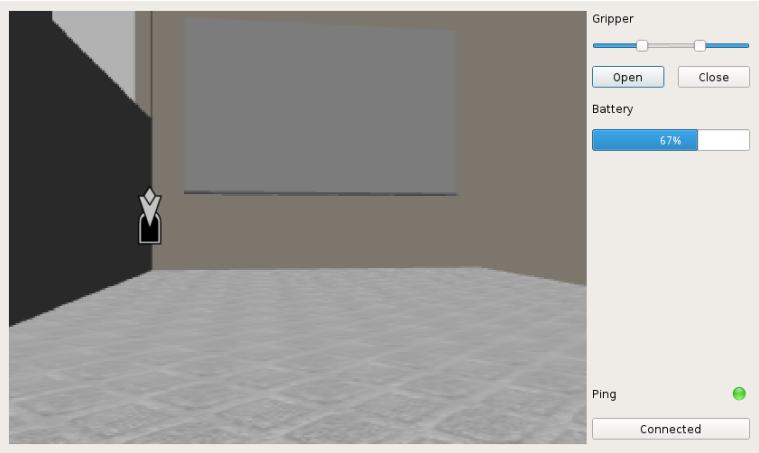


Figure 3.4: Mockup of an interface with a goal point indicator

which are the same driving distance from the exit. Using this setup, a usability study of the two interfaces can be performed to evaluate the interaction. If the interaction is deemed to be beneficial for the task, the results should be published and the connection between the game interaction and the navigation wayfinding task is enhanced with the information, providing evidence for use of the **goal point indicator** interaction in robot interfaces to complete a task generalized into a wayfinding task.

3.7 Conclusion

The Game Interaction to Robotics framework introduced here is a useful tool to create robotic interface improvements derived from game interactions. This framework can be used repeatedly and iteratively to accomplish two goals: completing interface improvements to robot interfaces, and providing useful research for others to use when they build new interfaces. The framework guides a robotic interface designer to choose a game interaction by first decomposing a specific task into a common task, and then guides choosing a game interaction based on the category of the common task. By parameterizing and describing the game interaction using a common game interaction model, a thorough understanding of the requirements is gained before mapping the elements to a specific robot interface improvement. This mapping aids in implementing an integration of the game interaction on an improved interface, with an attempt to

preserve the benefits inherent to the original game interaction. Finally, the improved interface is tested using well-known evaluation methods and standard robot interface metrics guided from the original task category.

These framework stages can be used and iterated upon for different game interactions and for separate robotic interfaces. Research done describing interactions in the second stage of the framework is not specific to any particular robotic platform, as interaction discovery is guided from the common robotic tasks. Game interactions are not represented in a significant portion of robot interfaces today. A significant number of interfaces from a long history of game development can now be used to enhance and improve robot interfaces. To provide a useful starting set of game interactions, a set of games is examined and some common game interactions are identified using the framework next in Chapter 4.

Chapter 4

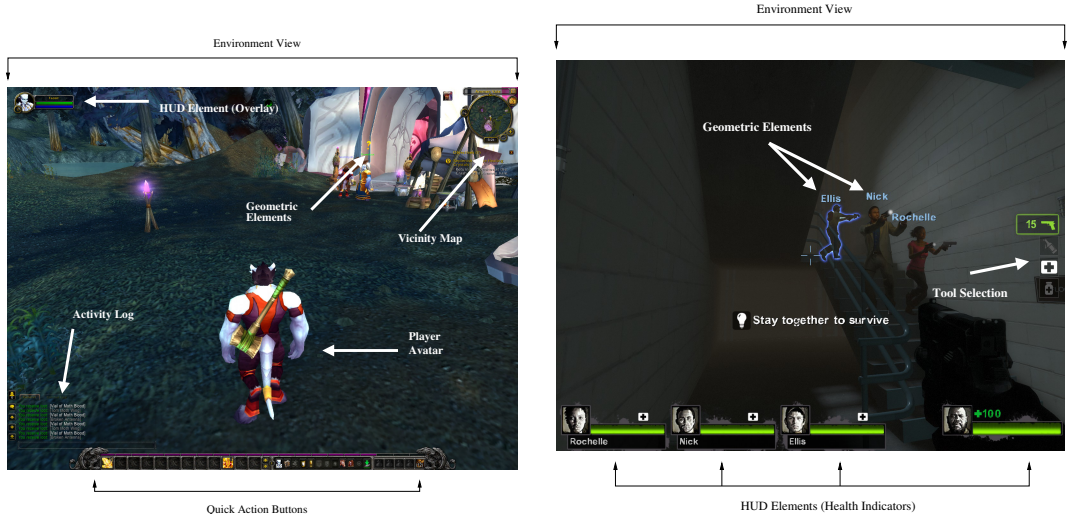
Common Game Interactions

Since the beginning of modern computing, video games have been created and enjoyed by millions of people. As they gained in popularity and variety, genres of games started to separate out into different types of gameplay based on narrative, interaction style, or viewpoint. These different types of gameplay experimented with types of interaction which would produce the most satisfying experience for the player of the game, with the goal to make the game as enjoyable as possible. As the gaming matured and gained even more popularity, a number of common methods of interacting with the game have emerged as a kind of expected interface style for games.

These game interfaces that share common controls and feedback related to the same action should be considered as the initial key examples of the Game Interaction Framework. Using the framework on these reused interface elements allow the most familiar game interactions to be transferred into robotic interfaces. A survey of modern games has been done to identify a variety of examples focusing on transferability to robotic interaction methods using first two stages of the framework. Not all game interactions will be advantageous, and failures of the heuristics which are identified from the survey have also been included as examples of interactions which either are impossible to transfer or do not fit with robotic interaction. Additionally there are some collections of interactions which represent tutorials or specific interface technologies which should be considered for use in robotic interfaces but represent an indistinct set of game interactions.

4.1 Game Survey Method

Within human-robot interaction and specifically examining mobile robot interfaces, two dominant styles of interaction are prevalent. Teleoperation is common when controlling a single robot, and provides for fine control of a robot with less of a focus on robot autonomy. Supervisory control methods, by contrast, rely on robot autonomy to free the operator for other tasks. A majority of games provide control schemes which can be consistently analogous to one of these two types of control methods: games where the player controls a single avatar, or games where the player commands a team of units in a supervisory manner.



(a) Third-person perspective

(b) First-person perspective

Figure 4.1: Notated screen of typical avatar-based games.

Controlling a single avatar was more common than the team of units. The term *avatar-based* will be used when the player is controlling or identifying with a single avatar. These were further split into first-person and third-person perspective interfaces. Annotated examples of both types of game screens are shown in Figure 4.1. Causing effects on the world in avatar-based games usually means selecting to execute one of only a few predetermined actions, then performed by the avatar that you are controlling on screen. A robotic equivalent would be choosing one of several autonomous actions, such as collecting a sample, or taking a snapshot using a video camera.

Figure 4.2: Notated screen of a typical commander-based game.

The term *commander-based* is used to describe when more than one unit is being controlled. Commander-based games contrast strongly in control style from avatar-based games. An example of a typical commander-based screen is annotated in Figure 4.2. In the commander-based games, units are controlled by you more abstractly as the nameless controller of a number of actors. As there are normally a large number of units to be directed, this means that interactions are often modal, requiring a unit. One consequence of this is that many of the interactions in commander-based games have preconditions. They present a number of units which can be directed to perform semi-automated tasks or autonomous tasks. Excellent support for heterogeneous units was also prevalent in the commander-based games. Exploration of the environment is a high priority in the game, with the revealing of a shared map being a key factor to success. These similarities hint that mapping interactions from commander-based games to supervisory control interfaces will provide improvement.

The genre of game also provided a useful point of reference when comparing games, as some genres shared a set of interactions within a category. One reason is that these genres are used when designing a game as players and creators identify with a shared set of rules and interactions. Building a game based on the expected interactions in

these genres allows the creator to short-circuit some of the "training" section of a game by assuming the player has learned how to play from previous games. A number of different ways to sort games into genres exist; Genre classifications based on the ones identified by Rollings and Adams[10] are used here.

Action These games are fast-based and normally require focus on reaction time and performance of the player. Typically there is one driving purpose behind each action. Usually the goals are simple such as traversing the level or defeating all enemies. A significant sub genre of this category is represented by the first-person shooter or FPS. In the first-person shooter, almost all interactions control an avatar which includes the camera, giving the namesake first person view. In many first-person shooters there are a variety of weapons which are available to destroy, impede or evade enemies. These are normally avatar-based games.

Strategy Typically played from an overhead perspective, they focus on generating structures or improving cities to produce units that will further the goal of the specific scenario. Two distinct sub-genres are separated by how the world clock runs. In real-time strategy, the world clock continuously runs, and time which is spent on a specific task cannot be regained, which means completing tasks more efficiently can be a major advantage. Contrasting this are turn-based games, where time is essentially stopped while actions can be considered without consequence. Most games in this genre are commander-based.

Adventure Completing a plot through exploration or interaction with the environment or non-player avatars in the world exemplifies this genre. These are only nominally distinct from role-playing games described below in the amount of rigidity that the storyline exhibits upon the character which is controlled. The adventure genre is dominated by avatar-based games.

Role-playing In these games, the player explores a world and advances through an environment, in a more open and less strict manner. A key element of the role-playing genre is the customization of the controlled characters, which gain abilities of the player's preference at distinct advancement points. A number of elements from role-playing games have been mixed into other game genres in recent popular

games. Both avatar and commander-based games exist in this genre.

Vehicle Simulation Typically racing or flying simulation games, the player takes on the role of the pilot or driver of a vehicle. Simulation games can be found for almost any vehicle, ranging from a space shuttle [48] to the subway train[49]. They also vary in simulation fidelity, from cartoonish racing games with powerups and no relation to real physics to the highly accurate simulators which are be used to train pilots. All of these are avatar-based games.

World Simulation With close ties to strategy games, these games are usually focused on simulating interactions of actors and the world at a particular abstraction level. In some cases they do not provide any specific goal, instead relying on the player's exploration of the simulation or internal goals. These games are commander-based.

Puzzle These games are atypical of the standard game, as they are usually a test of the player's mental acuity or planning skills. The player solves a set of puzzles based on rules set by the game. The goal is for the player to complete the puzzle using reasoning, logic, planning or experimentation. The gameplay can be "skinned" to produce games with similar premise and rules but a variety of different story lines. They can be considered as outside the avatar-based / commander-based system.

Some games mix a number of these genres, either by including sections which conform to specific genres, or combining elements from two or more genres into the same game. Both of these types of genre-mixing have been increasing in recent years. Mass Effect 3[50] for example is an action game where you control a first-person shooting avatar but also can select which team members to accompany you, contains sections where you pilot a vehicle, and also contains a limited amount of character specialization when characters advance typical of a role-playing game.

To discover common game interactions, a sampling of twenty recently released games were examined to discover interactions that were analogous or beneficial to common robot tasks. Games which were well-reviewed by video game critics were preferred, as they should provide the best interfaces with the least amount of issues. The set of games was chosen to span across multiple genres and have a near even number of avatar-based vs. commander-based games. The full list of games included in the survey are listed in

Table 4.2.

When examining games looking for common interactions, each game was played for at least 10 minutes. Notes were taken on all feedback. Screen captures were used to aid in identifying graphical elements and effects displayed on the screen. For most games a "main" screen view was identified which represents a majority of the interaction time with the game and interactions involving this screen were prioritized. In a minority of games surveyed, two screens were nearly equally prevalent and both were considered. Auditory and haptic feedback was also connected to each interaction. Finally, input from the player was observed. When a method to customize input was provided, the default setup was used.

After examination of the games for interactions, the notes were compared to identify widgets, view types, visual effects, and other interaction methods shared between many games. Interactions spanning multiple games were generalized to summarize the common elements. Each of the interactions identified was related to one or more of the common task categories. This represents significant original research, although it does not imply that the interactions are beneficial to all robot interfaces. Table 4.3 represents the initial matrix to be used when choosing an interaction to implement within a robotic system in the Game Interaction to Robotics Framework detailed in Chapter 3.

4.2 Interaction Details

Each of the initial interactions found in the game survey is detailed in this section, organized based on the common robot task which they beneficial to implement. Some interactions can be beneficial to a combination of multiple tasks as shown in Table 4.3, which are described in the primary task indicated, but referenced from the other related task sections.

For each interaction, the basic details are explained as they relate to the game interface and tasks completed, and the source games are identified by genre and view type. The benefits of the interaction relating to the common task are then presented. Next the second step of the framework is completed, parameterizing the interaction into the model as presented in Section 3.2, listing the input required, related objects, processing, and feedback presented.

Title	Release	Genre	Type	Sales[a]	Metascore[b]
Mass Effect 3	2012	Action	Avatar	4.83	89
Half Life 2: Episode 2	2007	Action	Avatar	2.82	90
Mirror's Edge	2008	Action	Avatar	2.17	81
Left 4 Dead 2	2009	Action	Avatar	4.01	89
Starcraft II: Wings of Liberty	2010	Strategy	Commander	4.29	93
Civilization V	2010	Strategy	Commander	1.27	85
Supreme Commander 2	2010	Strategy	Commander	0.47	86
Assassin's Creed IV	2013	Adventure	Avatar	1.92	88
The Legend of Zelda: Twilight Princess	2006	Adventure	Avatar	8.42	95
The Walking Dead	2012	Adventure	Avatar	1.42	92
World of Warcraft	2012	Role-playing	Avatar	10.01	93
The Elder Scrolls V: Skyrim	2011	Role-playing	Avatar	15.44	94
Diablo III	2013	Role-playing	Commander	4.19	88
Grand Theft Auto V	2013	Vehicle Sim.	Avatar	25.22	97
Forza Motorsport 5	2013	Vehicle Sim.	Avatar	N/A	82
Kerbal Space Program	2013	Vehicle Sim.	Commander	N/A	N/A
The Sims 3	2010	World Sim.	Commander	9.1	86
Simcity (2013)	2013	World Sim.	Commander	0.91	64
Tropico 4	2011	World Sim.	Commander	0.47	78
Portal 2	2011	Adventure	Avatar	4.04	95

Table 4.2: Games included in the survey.

[a]Global sales estimated by vgchartz.com, in millions of units, retrieved Nov 2013

[b]Reviewer score from metacritic.com. Scores range from 0 (universally disliked) to 100 (universal acclaim), retrieved Nov 2013

	Navigation						
Interaction \ **Common Task**	Localizing	Wayfinding	Movement	Perception	Management	Manipulation	Social
Minimap / Vicinity Map	●	○					
Fog of War	●	○		○			
Goal Point Indicator		●		○			
History Trail	○	●					
Joystick-based Avatar Movement			●				
Keyboard & Mouse Avatar Movement			●				
Simple Unit Movement		○	●		○		
Formation Movement			●		○		
Maximized Environment View				●			
Translucent HUD Element				●			
Commander-based Camera Movement				●			
Spatial Object Indicators				●			
Activation/Cooldown Time				●		○	
Health Indicator				○	●		
Activity Log					●		○
Unit Selection					●		
Group Selection					●		
Idle Unit Selection					●		
Quick Teams					●		
Action Queue					●		
Non-Modal Triggers						●	
Modal Activators						●	
Quick Action Buttons						●	
Tool Selection						●	
Unit-specific Actions					○	●	
Modal Default Unit Actions					○	●	
Multiplayer Ping				○			●
Multiple-Choice Dialog							●

Table 4.3: Matrix relating discovered interactions to common robotic tasks.
Filled circles indicate the primary task, with secondary tasks noted with open circles.

In some of the games, interactions were configurable, such that you could change the input actions which would trigger a specific interaction to occur in the game world. In game terms, the action is said to be *bound* to it's input. This concept splits the monolithic game interaction into two separate parts, the trigger and the action. The trigger includes the input, and some subset of the involved objects, targeted for the action, and any preconditions. The action uses the trigger object and the rest of the objects and performs a specific set of processing, and can provide a set of feedback specific to that action, indicating that the action has been performed. The set of actions which is available varied between games, but they are all could respond to any trigger they were assigned to. In practice this does not actually modify the game interaction for the purpose of the framework, as the final interaction mapped includes the action. For the survey interactions to be as reusable as possible, the trigger and action are noted when the game interaction can be configured.

When describing these game interactions, the mapped robot action will be non-specific, because it can be mapped to any suitable autonomous action. When specifying the mappings for the interactions which perform a non-specific action, the unspecified actions for the controlled robots can be drawn from a list of actions which can be started using a simple message or command sent to them and will proceed and provide feedback necessary to complete the interaction. This allows the interactions for manipulation or other actions to be translated smoothly to a number of robotic platforms.

Each interaction is put through a generic version of mapping to a robotic system to show the method of connecting the interaction elements to robotic interface and algorithms. Each object is coupled to either a robot, referenced algorithm to be used, or discarded. Details of moving the graphical feedback onto the new interface are also proposed. As generic targets, a teleoperation and supervisory control systems are used.

The teleoperation system is assumed to control a single mobile ground robot, using a joystick which drives forward and back, left and right in a differential drive fashion. The minimal requirements for the equipment of the robot only include a drive camera pointing in the direction of the forward motion which is angled to see the ground plane. This camera view is shown on the interface, and no other information is assumed to be transmitted or displayed.

Commander-based games present interactions which have some similarity to current

supervisory control interfaces in robotics. The generic supervisory system controls multiple ground robots which have sensors sufficient to localize to each other and produce a shared map using a multi-robot simultaneous localization and mapping algorithm such as that detailed in [51]. Robots can be driven using keyboard controls after selecting the robot to control on the interface. Each robot can also be given commands through a menu of actions available. A user interface mockup of this generic supervisory system is shown in Figure 4.3.

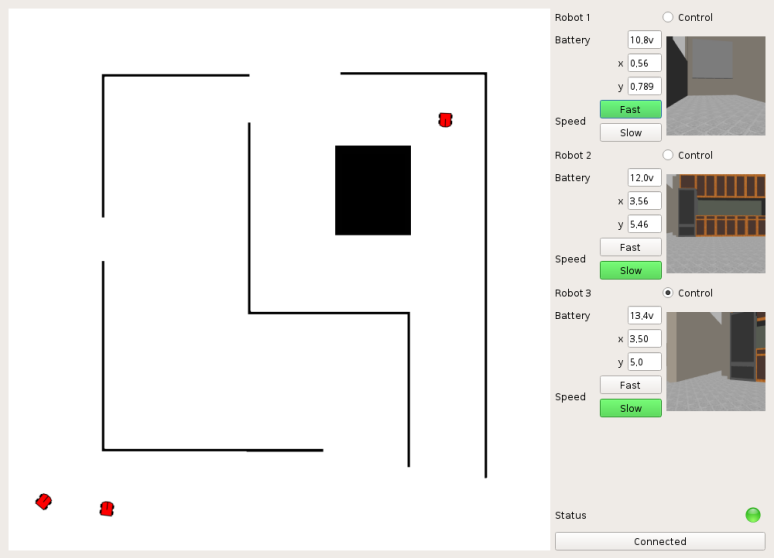


Figure 4.3: Mockup of a supervisory robot control interface.

One requirement to enable the interface as displayed here is a method to communicate and control many robots at once. Multiple recent robotics middleware can enable this type of interface and control, although some require a centralized intermediary. Using them, robots advertise their services to the middleware which aggregates them and facilitates connections from a client for controls. A method for determining available actions and other status will need to be standardized. The information required for the initial generic system consists of global frame localization, which enables displaying a unit that can be selected with for further interaction. The proposed interface connects to all available resources with the goal to provide a global view of units available.

When other robot sensors, algorithms, or interface enhancements are required on the target system to make the interaction feasible to implement, the individual interaction

description includes those details. A minimal set of additions are suggested to allow the interface to achieve the task, and more sensors or algorithms will be recommended to implement the interaction. When an interaction compliments, enhances or normally appeared with another, that information is also included as a possible direction for the next interaction to test in the next iteration of improvement for the interface.

4.3 Navigation

Navigation is a major task in robotics, comprising of three related subtasks: localization, wayfinding, and movement. Navigation was a common theme across many games, as it is a major part of most video games storyline through either explicit goal approach or open-world exploration. The History Trail interaction is another navigation-based interaction, discovery of which is detailed in Chapter 10 where the entire framework including a user study evaluation is used. Formation movement is similarly explained in Chapter 8.

Minimaps and vicinity maps

Miniature maps or vicinity maps are common in game interfaces. Of the games surveyed, 12 contained a persistent miniature map or vicinity map. Both types of maps are commonly referred to as a "minimap" by players. The **minimap** is a overview of the entire environment available for gameplay. Commander-based games are likely to have a minimap which will display the whole level or the discovered game world; Some examples from the survey are shown in Figure 4.4. Dots or icons on the miniature map represent objects or units in that area, with the color indicating which are owned by one player or team. An indicator on the map shows what section of the world is being currently shown in the environment view. The miniature map sprite updates in real-time with new information for locating assets on the field or enemy movement in vision of friendly units.

Avatar-based games have a similar indicator showing information in a vicinity immediately around the player's avatar location. However they may have details including icons for fiend or foe units or the location of objects key to the narrative that are not visible in the main view. In some cases there is a goal indicator shown with a directional

(a) Supreme Commander 2 (b) Civilization V (c) Tropico 4

Figure 4.4: Minimaps in games.

(a) Mass Effect 2 (b) World of Warcraft

Figure 4.5: Vicinity maps in games.

indicator on the edge of the vicinity map, or an icon if it is within the range of the map's radius. Two vicinity map examples are shown in Figure 4.5.

Both vicinity maps and miniature maps are used for a navigational purpose. The player in an avatar-based game uses a vicinity map to navigate the avatar they are controlling to a goal point around obstacles, or to localize themselves against known landmarks within the map. In the minimap's case, the map is used to localize the units on the screen with respect to the rest of the environment, or see the path controlled units will have to travel in the environment to reach a goal point. Having these maps available increases situational awareness, informing the player of location information or map information which would not be available otherwise.

The basic **minimap** interaction has no inputs, as it only displays information available from the game world. Many game state objects are used in the interaction. The level is one of the major objects - a miniature version of it is shown on the background of the minimap. The locations of friendly units are shown as dots on top, meaning that

the friendly units' position is another set of objects used. Finally, the current camera frustum's intersection with the environment is shown as a polygon on the map. This set of objects is processed by calculating the positions of the elements on the miniature map, which shows within it's boundaries the entire explored world. The feedback is graphical, and consists of the sprite rendering of the map background along with the dots representing units and camera frustum polygon on a drawing surface. The processing involved is the creation of this sprite rendering using the properties of the level, the positions of visible units, and the position of the camera. As units are localized with respect to the level map, adding the dots to a rendered version of the map is a simple scaling calculation of the coordinates to fit within the small sprite.

To map these elements to a generic robotic system, the level can be mapped to a shared map which is generated using a mapping algorithm. Care must be taken to globally align this map so that it can be represented by a single computed object. Using a multi-robot simultaneous localization and mapping algorithm would allow this map to be generated along with localization data to provide analogous objects for the friendly unit positions, or another relative localization algorithm can be used. The camera frustum would map to the environment view that is shown in the robotic interface - in the case of a supervisory interface this is fairly simple as the camera is positioned independently of the robots being controlled. Alternately if an view of the working environment is provided by a sensor, the pose of that sensor needs to be available to the interface to draw the frustum. Teleoperative interfaces can show the field of view of the active camera, or omit the frustum indication.

The **vicinity map** is a similar interaction, but uses a subset of the objects of the **minimap** interaction, consisting of only the portion of the level and units which are within a specified radius of the player's avatar. Extra processing also centers the map on the location of the avatar. In some cases, an extra step of processing was done to orient the map such that the direction of travel for the avatar was always "up" on the screen map. Some vicinity maps did not show the environment as a background as well, acting more like a radar than a map, in which case the level is not involved in the interaction. The feedback is the graphical sprite created using the processing, showing the portion of a miniature map which is within a specified radius of the player's avatar position. The mappings to robot systems follows from the **minimap** interaction. The

subset of objects and environment data required can make the **vicinity map** possible to implement on systems which have a more limited set of data.

Fog of war

Closely related to the **minimap** interaction is the **fog of war**, an indicator of explored area and information liveness. In all strategy games surveyed, a majority of each level map was unexplored and covered in an opaque overlay at the start of the game. As the level was explored by units under the player's control, the positions and type of the units determined how much of the level was revealed. Areas of a level with units nearby show up-to-date information continually. If the units moved away from the area, the most recent information about the area is shown with a translucent overlay. This indicated that the area was not being actively updated. The term "fog of war", alluding to the imperfect information in warfare, is widely used to describe the unexplored and explored but not updated sections, and the grey graphical effect reinforces the terminology.

Most games in this genre incorporate asymmetric information as a game element, requiring exploration of the environment to reveal the other players or events required to complete each scenario. This introduces an additional level of information availability, with the three states being unexplored, explored, and active. Areas of the world within a unit's visibility are active and are updated as new information is available. Unexplored sections of the map are covered in black, and the explored not actively monitored areas are grayed out or blurred. These explored areas showed a subset of objects which had fixed properties: one game only showed the level map, not mobile units or buildings. Another showed buildings but did not show the changes to buildings since the last unit visibility update. The **fog of war** combined with the minimap, showing a similar level map.

The interaction aids navigation and augments exploration in unknown environments, which is a combination of navigation and perception tasks. The fog overlay makes it easy to see which portions of a level have been explored and navigate correctly through the explored area. This means it is easy to direct units to explore the previously covered portions of the map. Areas of the map which should be monitored also are clearly indicated as being continually updated.

The **fog of war** has no inputs to perform it's interaction. Any objects which provide map vision to the player, which are mostly controlled units, the geometry of the environment needed to calculate the visible areas, and an object representing the exploratory state of the areas of the map. This object is updated by the processing in this interaction, which uses the visibility information and location of each object providing vision to update which sections of the map are visible, and mark sections not visible as explored. The feedback provided is graphical: an opaque overlay on the level area which is unexplored, and a translucent effect on all sections of the map which are explored but not visible. This overlay is applied to both the main environment view and the minimap if that interaction is present. All units which are located in visible locations are also revealed for updating the display on the minimap and the main view, although it is somewhat peripheral to this interaction.

Mapping these to our example supervisory robotic system, the objects providing visibility are the robots being controlled or other sensors contributing information to the map. The actively updated area from each of these sensors should be determined by their type. The fog object is represented by an occupancy matrix which contains three states: unrevealed, visible, and explored. First all matrix elements which were previously marked visible are marked as explored, and then the parts of the map which are within sensor range are marked as visible. The fog matrix is then combined with the map data displayed on the screen to produce the fog graphical feedback on the screen.

Goal Point Indicator

Another navigational aid provided in many of the explored games was the **goal point indicator**. This indicator showed spatially the location of the current selected goal in the world in relation to the avatar. The graphical representation varies, some games used an icon related to the goal type, and others used a generic icon for all goals. The indicator is placed spatially in the world but is strictly non-diegetic, existing only as an aid to the player. This interaction aids wayfinding by providing a general orientation and direction to the destination while performing a navigation task. It also enhances the perception of distance to the destination, through the magnitude of drifting which occurs when the avatar moves toward the goal; when the destination is far, it will drift less when the vector of travel is off-target.

In one game, the feedback was located spatially in the world if the goal was in the main view, but it was also visible on a compass element on the screen showing the direction to travel. In a different game, it worked together with an icon on the edge of the **vicinity map** to show the direction. Most examples came from avatar-based games in the role-playing and adventure genre, although one world simulation game with a commander-based view also used them in the tutorial levels. The indicator was either automatically activated by a trigger in the game engine, or selected by the player from a auxiliary UI showing a map or quest list.

The **goal point indicator** interaction is activated in various different ways in the UI or automatically by the game. If activated by the player, the commands to activate could be considered input but once entered, no additional input is required. Alternately the existence of a goal point could be considered a prerequisite for the interaction. The objects involved are the goal object, and the main view camera pose, which was related to the avatar pose. Processing calculates the location of the graphical indicator in the game world if visible, or the location of the directional marker if not visible. It is based on the pose of the camera and the vector between the camera and the goal object. The feedback elements generated is a geometric indicator located in the world or an overlay sprite showing the direction, depending on whether the processing produces an item on screen or

To map the interaction onto our example teleoperation robotic system, the avatar position and associated driving camera are mapped to the teleoperated robot and driving camera. The goal object must map to a real-world coordinate either selected through the interface, or specified beforehand as a point of interest. The vector between the goal point and current point is required, so a localization algorithm must be employed, accurate odometry and a goal point in the robot reference frame, or other sensor providing this vector is required. Once this vector is provided, the feedback can be mapped directly on the environment view using an augmented reality overlay to place the indicator spatially, or added to a HUD element which can be added.

Joystick-based Avatar Movement

Avatar-based games mostly offered interactions for movement using a console gamepad. All surveyed games provided a similar interaction for moving the player's avatar and the

camera, with minor variations between each game. Control is split across the two analog joystick controls, with the left joystick controlling avatar movement in the forwards and backwards direction and side-stepping, and the right controlling avatar rotation and tilt of the camera. Each axis is interpreted as a percentage of effort: when the joystick is at its limit, movement is at it's maximum, with increasing response as the joystick moves from the resting point. In most games using this interaction, freedom of movement consisted mostly of movement in a single plane, with vertical movement occurring in certain areas based on the environment. The same interaction was used in both first and third-person perspective avatar-based games.

Avatar-based games with a third-person perspective sometimes contained some additional movement of the camera as well. When the avatar is not moving through the world, the camera can be controlled freely as described above, allowing tilting and rotation. Once the player moves the character in the world, the camera returns to a view situated behind the avatar. In addition, when the movement is significantly upwards or downwards, including ladders and vines for vertical climb, the camera view adjusts to display more of the environment in the direction of movement.

The input for this **joystick avatar movement** interaction is the axis on the two analog joysticks. Objects involved are the camera within the simulated world, the player's avatar. The processing is the translation and rotation of the camera and/or the avatar in the world based on the rules explained above. Processing can also restrict movement of the player's avatar and the camera based on the game environment and rules, in which case the object for the environment must be included to avoid collisions. One feedback element in this interaction is relayed graphically through the main camera's view change shown on the screen. Feedback is also provided in the form of animation of the avatar on screen if it is visible, and possible auditory feedback in the form of walking or running sound effects.

The mapping to robotics is straightforward, with the addition of a gamepad to the robotic interface the joystick can be mapped directly. The player's avatar is mapped to the robot being teleoperated, and the feedback into the driving camera which should be transmitted. The basic processing is translated based on the robot being controlled. The example teleoperative robot is non-holonomic, so the horizontal axis of the left joystick does not have a meaningful translation. It can be simply ignored instead.

The rest of the processing translates reasonably, with the vertical axis of the joystick moving forward and backwards, and the right joystick turning the robot in place and controlling tilt of the camera. An operator who is familiar with the controls from the game will still have familiarity, but restricted to possible movement. The camera movement feedback naturally occurs due to movement of the robot platform. Auditory feedback can be added to the interface to indicate that commands are being sent or that the robot is moving, although the selection of footfall sounds may need to be replaced. In recent years, some robot control interfaces have used a gamepad, as a convenient, easily available, familiar and well-tested input device, such as the ones seen in Figure 4.6.

(a) Packbot Controller (b) TALON Controller

Figure 4.6: Robot controllers with game pad designs.

The more complex interaction with the automatic camera centering and tilt can also be parameterized as an enhanced version of the interaction. There are no additional inputs, and an additional object representing the nearby environmental terrain is used. For processing, the game engine checks if the avatar's movement is proceeding in a vertical direction or if the local terrain in front of the avatar is rising or falling, the pose of the camera is adjusted up or down accordingly. The feedback is unchanged from the basic version.

To map the interaction to a robotic system, the mappings from the basic version continue unchanged, and with the player's avatar mapped to the teleoperated robot, the camera to the driving camera, and the view to the feed from that camera. The description of the local environment can be mapped to a laser scan which would provide an estimate of positive slope amount or that a neutral or negative slope exists, or possibly a horizon detection algorithm could analyze the camera feed. The processing

involves the progression of position of the avatar, so a method of retrieving the change in location must be provided. This can be provided via an accelerometer or by more a more complex localization. The change in tilt would be governed by a combination of this movement sense and the slope detection algorithm, with positive slopes biasing tilt up.

Keyboard and Mouse Avatar Movement

Avatar-based games that were available on the PC platform provided a method for avatar movement by combining the keyboard and mouse inputs. Many provided both **joystick avatar movement** if a gamepad was present and **keyboard and mouse avatar movement** interactions. Control is split between the keyboard and the mouse, with the keyboard controlling movement analogous to the left joystick from the **joystick avatar movement** interaction, with the mouse controlling rotation and camera tilt. Four keys in a "inverted T" positioning on the keyboard (typically "WASD" keys) are assigned for movement forward, backward, sidestep left and sidestep right. Keys are binary, so the movement is usually ramped to 100% of speed in a direction over a period of time, and sometimes has a similar deceleration ramp when released. The mouse replaces the joystick by acting in a mode referred to as "mouse-look', where relative movement in the vertical direction tilts the camera up and down, and horizontal movement rotates the camera and/or avatar in the direction of movement. They are absolute movements in contrast to the percent effort used by the joystick method.

Input for this interaction is composed of the keyboard key and mouse position input. The involved objects are the player's avatar and the camera for the main view. The processing interprets the keyboard keys pressed and the mouse movement and updates the pose of the avatar and camera as described. Feedback mirrors the joystick based movement interaction, with the same avatar animation being used.

The mapping is again fairly straightforward as in the last section, with similar caveats, but an additional complication related to the mouse movement and reaction time. While the effort-based movement works well for the joystick movement, the "mouse-look" moves as quickly as the mouse can be moved across the surface it is resting on. As the speed of the robot rotation is physically limited, in the game interaction, there is often no limit on the speed of rotating. In robotic systems the limit on

the rotation and tilt of the camera is likely to be quite slow. This means the response will either lag behind mouse movement, or need to have an upper limit low enough to be frustrating for the operator moving the mouse. This limitation will need to be communicated to the operator in some way or worked around.

Simple Unit Movement

Unit movement in commander-based games is a basic task that must be performed thousands of times within a single level. Exploration is a common task in commander-based games - the player which has better information about their opponent's structures and strategy and the available resources is better equipped to direct their units to accomplish their goals. The primary method for exploration in the games is by moving units around the map, revealing information as they travel. All commander-based games encountered assigned the secondary button of the mouse to a **simple unit movement** interaction and it was the most common interaction used. It was activated by simply clicking the secondary mouse button at an otherwise empty target location while a unit is selected.

This navigation interaction also involves wayfinding, as the game engine plans traversable paths around obstacles to reach the target location. If the target location is unexplored, the path planning often switches to a method suitable to the unknown information. This type of planning represents a higher level of autonomy than the previously discussed methods of movement.

The interaction has a prerequisite of a selected unit with movement capabilities. Input for this interaction is the secondary mouse button click, at a specific screen location. The player-known map is used as an object along with the selected unit. The processing then uses the click location and the main view camera location to translates it into a map coordinate. The map coordinate is passed as a destination for the path planner with the starting point of the selected unit's position. The path planner outputs a set of waypoints which are followed in order by the selected unit. The graphical feedback consists of a spatial element indicating the destination location, attached to the commanded unit by a line along the ground. In some games there was auditory feedback indicating the acceptance of the movement task by the unit being moved.

Mapping the interaction into the example supervisory robotic interface, the selected

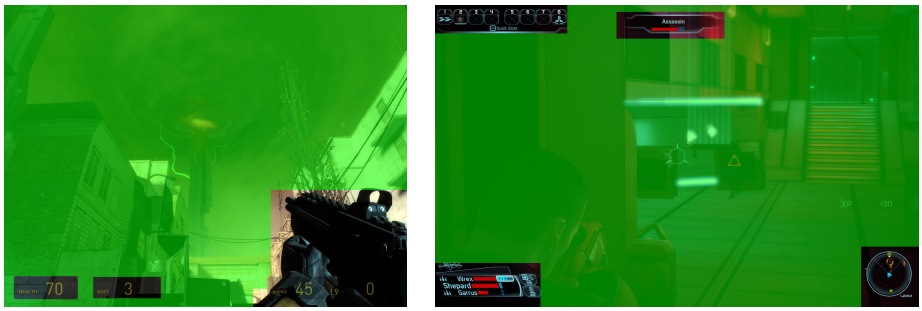
robot maps to a selected robot in the interface. The mouse input and camera positioning are mapped almost directly, with the mapping actually made simpler due to the 2D nature of the robot interface. The player-known map object is turned into the known map. Processing requires a path planner, of which there are a number of suitable algorithms available for robots. The location of the robot must be passed as the starting point to the path planner, so some localization is needed for the robots. Path planning can be mapped to be performed by the robot - in which case the robot is sent the target location in the correct coordinates. Alternately the interface could perform path planning and send waypoints to the robot, requiring less robot autonomy and monitoring it's progress. Graphical and auditory feedback can be mapped to similar indicators in the robot interface, possibly adding indication that the robot has received and is executing the planned movement.

4.4 Perception

A close second to navigation within games are tasks related to perception of the environment. As the environment is necessarily physically separated from the player of games, a number of interactions are focused on providing the player a concise view of the world that the avatar is operating within. Many of the game interactions discovered are static or dynamic display of game state, and take no input to relay their information.

Maximized Environment View

In almost every game examined, the screen is dominated by a view into the game world. In a majority, the entire screen was filled edge-to-edge with this view. Even in games with the smallest environment view, as the screen included other fixed elements, more than three-quarters of the screen is filled with the environment view. The display of the world in this way for first-person games increases immersion of the player in the game world presenting the game world similarly. The use of this maximized main view shows the immediate state of objects in the environment surrounding the character or within the immediate vicinity. In some sections of the game, it was the only feedback displayed on the screen. It is so ubiquitous that almost every screen could be used as an example for this interaction. In Figure 4.7, the non-occluded area occupied by the



(a) Half Life 2, 84%

(b) Mass Effect 2, 90%

Figure 4.7: Primary view dominates the screen - the primary view not occluded highlighted in green.

primary view in two games has been highlighted. As earlier games did not have this "edge-to-edge" primary view and almost all games in the survey of recent games do, it represents an evolution of the game interfaces to prefer the interaction for possibly competitive reasons.

Robotic interfaces often provide a feedback similar to this view, but in current systems usually a much smaller portion of the screen. Using the entire available display feedback area increases immersion in teleoperation robotic interfaces, and provides for more of the operating environment to be displayed simultaneously in supervisory interfaces. This increased view in the game is beneficial as it enhances the immersion experienced in the game, and makes it difficult to ignore the game world being presented. While it could be thought of as following human-computer interaction guidance to maximize the "working area", the designers of FPS games also consider more variables of this view such as the field of vision angle and variation of the view while movement (a.k.a. "Headbob").

This interaction requires no inputs. The primary object in this interaction is the camera object, including it's pose and rendering parameters. Indirectly involved are the environment object and any other units within the view frustum that must be rendered. Feedback is graphical, with the game engine rendering the environment and the units in view using an established model and (usually) with the aid of specialized hardware. The feedback for this interaction represents the majority of 3D rendering. The most effective versions of this interaction take over the whole screen or window with this

rendered view, and other user interaction feedback occludes this view.

To map the interaction to our teleoperative interface, the camera object is represented by the driving camera. Properties of the camera used for rendering like the field of view and resolution are replaced by the physical properties of the sensor. The rendering system used for feedback is functionally mapped to physics, transmitting the image from the robot camera to the operator interface. The feedback is different from the standard driving interface as it follows the properties of the game interaction and uses the entirety of the screen instead of only a section. The rest of the controls are made translucent or transparent to accommodate this.

Translucent HUD Element

As mentioned briefly in the **maximized environment view** interaction, many games have elements overlaid on top of the view, showing various indicators or status. These took the form of a translucent or transparent section of the screen with a numeric and/or pictorial indicator. Usually there are more than one of these **translucent HUD element** interactions on the screen at once. A player's health and armor indicators are shown in Figure 4.8(a). This interaction is present in commander-based games as well, with Figure 4.8(b) showing a set of available resource indicators. These HUD elements often provide information which would be unknown to the player otherwise. With the extra information the player might alter their decisions or understand the effects of the environment and interactions on the avatar they are controlling.

Considering all instances of the HUD element, general trends appear. Transparency allows the player to continue to see action occurring within the vicinity of the indicator. Many are minimized in size to reduce the area occluded, containing icons or abbreviations instead of text. Common game information relayed are numbers indicating the general health of the player, remaining ammo counts, the currently equipped weapon, other resources available, scores, or the time remaining in the level. In avatar-based games the elements tend to be specific to the unit, while in commander-based games more global information is usually provided. There are typically less than six HUD elements on the screen simultaneously. Some elements are shown transiently, only when something has been changed or they are relevant - for example, equipping weapons without ammo will hide the ammo count indicator. In some games, these elements could be

(a) Half Life 2 (b) Starcraft 2: Wings of Liberty

Figure 4.8: Examples of translucent HUD elements.

disabled as a group.

This game interaction is focused on presenting the user information about the state of the game world, making it a perception based interaction. The feedback is on screen constantly, requiring no input to show. As it is a much-reused and quite generic interaction, the objects involved depend largely on the type of data being displayed. At least one object provides data, so we can generically call it the source object. An optional icon is also provided, based on the source object and used to enhance the feedback. The feedback is represented by the HUD graphic on the screen, a translucent box with the number from the object alongside icon or text indicating what the number represents. Processing is trivial, or assumed to be completed before this interaction is used to display the results.

The mapping to either the teleoperation or supervisory interface is simple, with the feedback displayed on top of the video view in the case of teleoperation and in the environment view in supervisory interfaces. The feedback can be preserved, with minor changes to the font and/or the icon to represent the correct value. The source object could map to any number of datum available to the robot. Some examples could be wireless signal strength, current velocity, remaining battery, heading, or even something more abstract such as number of detected targets or vibration indicator. This interaction can also be duplicated for display of multiple objects. The location and number of the feedback elements are choices that can be guided by similar displays within the games. The relatively low number and the location on the periphery of the screen are good guidelines to follow.

Commander-based Camera Movement

In contrast to the avatar-based games where the camera moves only with movement of the avatar within the world, the camera in commander-based games is free to move about

(a) Supreme Commander 2 (b) Starcraft 2: Wings of Liberty

Figure 4.9: Typical views in commander-based games.

the environment. Moving the camera helps the player explore the area of the level which has been explored and know where units are positioned. The requirement to interact with multiple units in a tactical way is emphasized as well. In the commander-based games examined, the terrain is represented as a flat plane or 2.5 dimensional plane with different heights for each position. Separate planes for ground and "flying" units are used, providing limited three-dimensional positioning. The default camera view approximates an isometric view, with approximately a 45 degree angle to the plane. In two games included the player could adjust the view or rotate the camera freely and zoom. In practice it was not often used as it provides less information about the world state. In typical views, shown on the top in Figure 4.9, the camera has a view of a section of the plane. Units are placed and shown according to their position in the plane. It is not uncommon for hundreds of units to be shown in a single view. The camera view even when at wide zoom does not show the entire plane.

When the player moves the camera one of three interaction methods can be used. The first is activated by moving the pointer to the edge of the screen. This causes the camera to pan in the direction of the edge the cursor is at until the cursor is moved from the screen edge. The arrow keys on the keyboard are the second input method, which performs the same action as mouse edge movement without requiring the movement of the cursor. The third way is activated by clicking on the minimap on the screen. This moves the camera instantly to a location showing the represented location clicked.

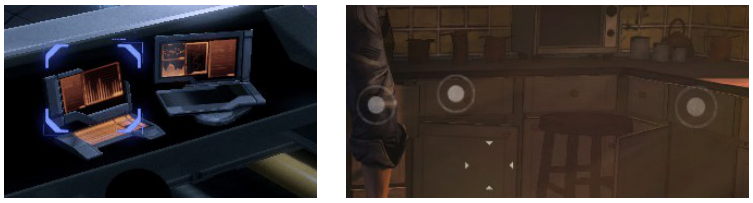
The first two camera movement interactions can be described together as they share processing and feedback. The input is the only difference. In the mouse interaction the input is the cursor positioned on the side of the screen. In the keyboard interaction a key is pressed. The only object involved is the camera object, which the processing moves at a constant speed in a direction based on the key pressed or the edge the cursor resides. Processing also checks if the camera edge is already aligned with the level edge, in which case nothing is done with the camera position. Feedback is presented by movement of the camera position rendering the main view.

The other interaction has as a prerequisite the use of the mini map interaction and display on the screen. The input is the click location of the mouse on the mini-map. The same camera object is modified. The processing takes the coordinates on the minimap and translates them to game world coordinates which are then mapped to a new camera position. Then it calculates the new position camera based on centering the location clicked, with the restrictions of the camera view within the environment similar to the keyboard and mouse-edge interactions.

To map these interactions to the example supervisory control interface. Inputs can be mapped directly, with the keyboard and mouse. The camera is mapped to the object controlling which section of the environment is showing in the main part of the interface. The processing can be mapped with the limits being based on the mission area. Feedback is presented by redrawing the interface to show the new parts of the area within the main view.

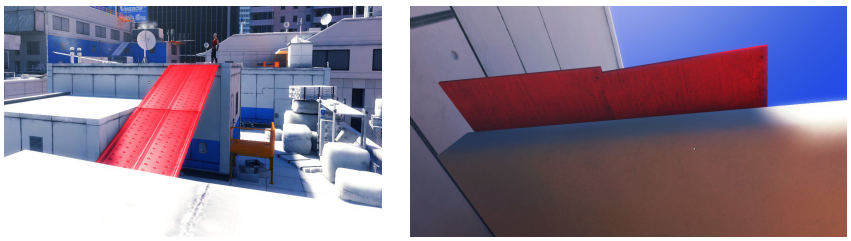
Spatial Object Indicators

Along with the **spatial goal indicators** discussed aiding navigation, other indicators also appear spatially and enhance the player's perception about the environment. Present in avatar-based games, this interaction shows a visual effect on specific objects such as a glowing halo or an graphics overlay. This indicates to the player that object is important or can be interacted with. In some cases these graphical enhancements are explained in the narrative as diegetic as future technology (an advanced HUD identifying objects). Examples of this interaction include floating text showing that an item can be used or picked up, a highlight revealing a teammate otherwise occluded by the environment, or player names above avatars. Several examples are shown in Figure 4.10.



(a) Mass Effect 2 (b) The Walking Dead

Figure 4.10: Examples of the **spatial object indicator** interaction.



(a) Boards to jump (b) Ledge to grab

Figure 4.11: Highlighted spatial objects in Mirror's Edge.

Figure 4.11 shows another variant of this interaction from Mirror's Edge[52] where game world elements are shaded with a different color. In some cases these spatial highlights are visible through obstacles, producing a type of "x-ray vision" for important objects.

When parameterizing the **spatial object indicator** interaction, no inputs are required. The object which is being highlighted as well along with associated information which is made available, and the camera view pose is used. The feedback is a graphical output which is geometrically located highlighting the object location. The exact representation of the feedback varies across games, but normally framed the object to be interacted with. In some implementations, text is displayed near the feedback. The geometric representation ignores the normal occlusion of the renderer. The only processing required for this interface is a calculation of where to place the indicator within the camera view pose, and a filter to only show the indicator when some conditions are met, such as the camera being within a certain distance of the object.

To map these to the teleoperation robotic interface, the main view camera pose is mapped to the teleoperated robot driving camera, and the object being highlighted by the indicator to an object determined to be manipulable by the robot, or another object

Figure 4.12: Activation time animation in World of Warcraft.

of interest in the view. The object of interest could be a particular fiducial, a recharging station, or another robot or human which should be brought to the attention of the operator. In this example, we choose a recharging station which is detected to be in range. The processing would then translate to calculating whether the charging station is within view based on the camera pose and the location of the charging station, and if it should be displayed based on the length between the camera pose and the station. The feedback is mapped to an indicator generated and overlaid on the main view using the projection of the 3D location of the charging station into the geometry of the camera pose.

Activation and Cooldown Time

Paired perception-based interactions based on timers are the concepts of **activation time** and **cooldown time**. **Activation time** is an interaction involving an amount of time a character must wait when starting an action, to complete the action. This time is represented by a progress bar or circle which slowly fills. When the progress bar completes, the action "fires" and the results occur in the game world. This progression can be seen in Figure 4.12. During this activation period, the player was unable to perform other actions or move. This set of disabled actions can be different depending on the action activating.

Cooldown time is a similar concept, but applies after the player has successfully completed an action. When an action has a cooldown time, it is unavailable after use for a specified amount of time. Uncommonly, a single action starts cooldown timers for a set of related actions. While this cooldown timer has disabled an action, an animation is shown similar to a clock wipe. The duration matches the timer, allowing the player to judge how short the timer is and notice when the action is available again. Combined, these two delay types are used by game designers to limit the use of strong actions while still making impressive actions available to the player.

Activation time is parameterized using a generic action being activated, which is one involved object. This action specifies the length of the timer and the set of disabled actions while activating. Another object represents the timer that is used by the interaction. An activation timer can be started multiple ways. One way is to use a **quick action button**, discussed in the manipulation section. The feedback is a graphical overlay showing the progress bar filling on the screen over a number of seconds. The beginning of this feedback coincides with the start of the timer by the processing. While the timer is running, the specified set of actions are disabled by the processing as well. When the timer completes, the actions are re-enabled and the effect of the action begins.

The **cooldown time** interaction takes no inputs, being started after the completion of an action. This action is again an involved object, specifying the length of time and the set of actions to disable. A timer object is involved as well. Feedback exists in the form of a translucent shade over disabled actions, slowly being removed in a clock wipe animation based on the time specified. In processing similar to the **activation time**, the actions are disabled, a timer is run and the actions re-enabled at the end of the timer.

A number of robot actions could benefit from the transfer of these game interactions. Any algorithm which requires the robot to be stationary while the sensors perform a long-running action could benefit from the enforced delay of **activation time**. One example is a moving object detection algorithm might require the robot to be stationary for a few seconds for analysis. An action that cannot be performed in rapid succession such as hardware that must perform a reset could use **cooldown time** to mediate the activation. Perhaps a compressed actuator can cause the robot to jump, but needs half a minute to re-compress once fired. One example using both would be creating a complex model of a room by a 3D sensor. The activation time would be used to keep the robot stationary while collecting the data, and the cooldown time used to ensure another scan is not captured while the map of the room is being synthesized.

Consider this room modelling example and mapping to the teleoperating robot platform with an additional 3D scanning sensor. The input of the **activation time** is mapped to a button on the keyboard to start an action. The feedback is a progress bar labelled “Mapping area”. The object for activation time specifies the amount of

time needed to complete an area scan and that movement is disabled during this time. Processing is translated to starting the scan in response to the button, and sending messages disabling the motors, and starting a timer to re-enable the motors at the end. When the scan completes, the motors are re-enabled, and the processing task effect is started.

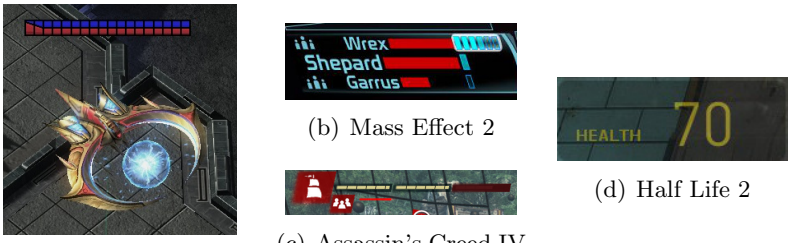
Continuing on to the **cooldown time**, the feedback is mapped to an element on the screen which shows the amount of time left in the timer before another scan can be started. This is represented on the screen as a overlay icon with a clock sweep. The processing reads the object representing the cooldown time associated with a map processing task, and the set of actions which includes starting an area scan and other things which use high amounts of processing are disabled. It also starts a timer to re-enable those actions at the end.

4.5 Management

Management as a robot task is defined by coordinating the actions of multiple robots, either acting as a group, or independently from each other. Multiple types of management are considered, including management where one or more of the entities are not under the control of the player. Interactions correlated to management tasks involving multiple units are well-represented within commander-based game interfaces. The other aspect of management is the internal status of controlled robots. Displays of internal status is one area where you expect to find analogous interactions within games.

Health Indicator

One interaction appearing in more than half of the games examined was the health bar or another **health indicator**. This indicator is represented on-screen using a bar which depletes at the player is damaged or within a damaging environment, or a numerical indicator. They are often but not always percentage-based, with 100 meaning “no damage”. In some cases the player can gain “more health” in which case the bar size is increased. With a bar indicator, the section which is missing is important as it shows the missing amount as a ratio to “full health”. A selection of health indicators from a variety of games is shown in Figure 4.13. This simplified representation provides a way



(a) Starcraft 2

(b) Mass Effect 2

(c) Assassin's Creed IV

(d) Half Life 2

Figure 4.13: Health indicators in games.

for the player to quickly evaluate a level of danger or detect damage as it happens. In commander-based games, the health indicators are were spatially presented near units.

There is no input for this interaction. The feedback is the graphical indication of the calculated health, sometimes presented as part of a **translucent HUD element**, but instead often is represented as one or two bars with segments that are filled or unfilled. The colors of these bars would sometimes change in reaction to the level presented, with green representing high health, yellow lower, and red being the lowest. In some commander games, the bars were only shown when the health of the unit was not full. The only involved object is the unit which the health attributes are being drawn from - typically the player's avatar in avatar-based games, and a specific unit in commander-based games. In some cases there are two different objects, one for unit health and another for a "shield". In the Mass Effect game, three of this interaction were presenting the three characters on the team (including the player's avatar). Processing is performed to translate the state of the unit's health into the correct single-axis data to display on the feedback element.

When mapping the interaction to a supervisory robotic interface, the object is mapped to a robot being controlled. The feedback is represented by a health bar located spatially near the iconic display of the robot. The processing does not translate easily, but we can replace the health state with something simpler to compute such as the amount of battery remaining. This interaction would then allow the operator to easily assess the battery level of all of the robots, seeing quickly the ones which need to be recharged without checking all individually.

Another mapping could be presented for a teleoperation interface. This mapping

would use the robot being driven as the object that health is derived from. The processing can be more complex, representing the general operational status of the robot. This composite metric would need to be developed using the state of all of its subsystems. This metric should be developed to have similar properties to a health indicator - if some failure is causing degraded performance, it should lower the number until repairs occur.

Activity Log

Appearing in only five of the surveyed games, the **Activity Log** interaction was less common, but interesting as one of few teleoperation-applicable management interactions. The **activity log** appears in games as a list of events which have occurred in the game world. In robot management the interaction should bringing attention to important events that require operator attention that might otherwise go unnoticed. A wide net is cast when determining what constitutes an event, which could be as trivial as one player scoring in a game, or as important as the game ending. The displayed log events inform the player about a large amount of activity occurring in distant areas. This information can be ignored when busy or checked later at the discretion of the player. In the typical implementation seen, any recent log entries are visible, until a maximum number is reached. The log can sometimes be expanded to inspect less recent entries. In commander-based games, the activity log entries may contain elements which can be clicked to move focus on the location of that event.

There are three interactions which are bundled in the **activity log**, which can be dealt with separately. First is the log display itself, which has no inputs but presents feedback of events. If the log is thought of as a kind of message board being maintained by the game engine, the set of involved objects for the interaction is limited to the log object. The engine processing determines which events are shown by filtering the log based on the current player's properties: their team, location or log timestamps. The associated feedback is the log text, which shows the set of events determined by the processing on the screen, and shows usually either in a corner of the screen or momentarily just below the center. When considering a mapping of this first interaction to a robotic interface, the object containing the log can be replaced by a blackboard system or event broadcasting system. Some middleware such as ROS have a log method

built in, in which case the existing log can be used. The processing can be simplified to only do time-based filtering, or a configurable filter. The feedback can be transferred closely as well, with possible modifications based on more complex information included in robotics events. One note for this interaction is that it is not specific to either teleoperation or supervisory interfaces.

The second game interaction which can be found is the ability to recall the entire log. As input, a keyboard command or mouse click on a virtual button on the UI is pressed, which then toggles visibility of a scrollable window of all log events as feedback. A fairly straight additional interaction to add to the previous one, the only object involved is the activity log. There is minimal processing, only the filtering of the global log as it is displayed. The mapping is trivial in this case, following from the previous interaction. The scrolling controls will need to be implemented. The only complication is the requirement to store more log entries, which may need to be done within the interface if the middleware interfaced does not have a method for retrieving historical log entries.

The last interaction involved with the **activity log** is the ability to interact with a log entry. This interaction was only found in commander-based games. The input in this interaction is a click from a pointing device on one of the log entries. The objects involved are the log entry itself, an object indicating a location associated with that log entry, and the camera object. If no location is associated to the log entry, then the interaction does nothing. Otherwise, the camera is moved to the position involved in the entry. In addition to the camera movement, the unit or position involved in the log is briefly highlighted geometrically. To move the interaction into a supervisory interface, consideration is made for interfaces which display the entire map. In these cases, the movement of the camera is not required, but an embellished highlighting can be provided. Processing may need to be modified in the robotic system to attach related objects and/or locations to the log entries as they are added to the log.

The activity log could be a major improvement when monitoring a large number of semi-autonomous robots in a supervisory interface. The robots can finish assigned tasks at different rates and at disparate work locations. In rescue situations where a large number of robots would be useful to search the area quickly, a overview containing a number of autonomously exploring robots with the activity log could be used to

Figure 4.14: Selected unit modal interface in Civilization V.

coordinate when events occur that require operator intervention.

Unit Selection

In all the commander-based games surveyed, the input used a keyboard and a pointing device. The pointing device's primary button defaulted to **unit selection** when used in the environment view. A unit is selected by clicking on it, which produces a highlight geometrically around that unit. Detailed information about the unit is then shown spatially on the map within the details panel and action panel. An example of the details interface which appears when a unit is selected is shown in Figure 4.14. In the details information about the selected unit, including health and armor status, upgrades, and an avatar representing the unit. The action panel (vertically on the left in the figure) shows orders enabled for the unit.

The input to this interaction is the primary button click. Objects involved include the location selected, the unit which is being selected, as well as the object within the game engine tracking the currently selected unit. Processing assigns the clicked unit as the selected unit, updates the status to enable showing details and available actions. The feedback consists of the geometric indicator showing that the unit is selected, which is a circle on the ground around the unit. The details panel of the screen changes to show the statistics of the unit, and the action panel updates showing the available actions.

An easy mapping into the supervisory system is to correlate the unit being selected with a robot on the interface, replacing the traditional radio-button already on that interface. The inputs, feedback, and the pointer object need not not change significantly to implement this interaction. Currently all robot details are shown on the right bar

would be removed, replaced with single detail section instead. This detail section will display the statistics of the robot, with more space and flexibility to show unique sensors. The tracking of the currently selected robot can be stored locally in the interface. The feedback would update the map and robot view, adding the circle around a selected robot.

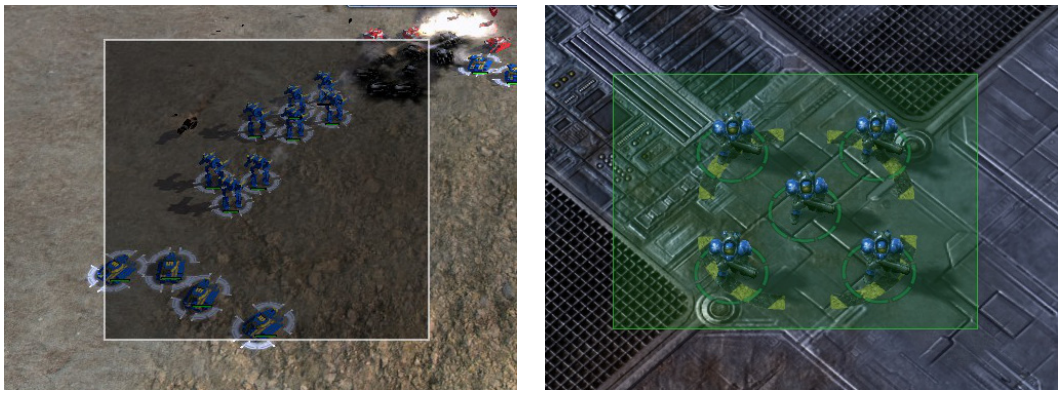
This type of selection enables an interaction which supports an overview of the working environment plus a detail view on a particular robot. The world represented in the view has too much information available to present for examination at once - given even four robots there would be not enough space on the current interface.

Group Selection

Commander-based games reviewed enabled selection of multiple units in many ways. To select by locality of position, the player uses a "rubber band" selection box similar to that used in spatial file management interfaces. Clicking and dragging a box starting at an unoccupied location selects any units which fall within the box. This can be seen in Figure 4.15 in two games. Units are often organized by players locating them near each other. This group selection method encourages this grouping; they can be easily selected en masse to assign a task to perform in formation and/or unison.

If the player would rather select all units of the same type, the method for selecting varied across games. In Starcraft 2[53], double clicking on a unit, instead of performing unit selection as discussed previously selects all units of the same type visible on screen. Another method used is to filter a larger group of selected units, by clicking on a unit's representation in the details panel while holding the Control modifier.

For the **group selection** interaction, the locality selection method will be described here. Some parts of this description can be reused when describing other methods that have similar inputs, feedback, or processes. Group selection is a fairly common interaction within commander-based strategy games, having multiple methods available within the same game interface is common. For locality-based selection, the input is the mouse drag: a location where the mouse is pressed and a second location where the mouse travelled to. These two locations are processed using the view camera geometry to produce points in the environment, which define a box in the world. Processing uses this box and returns units within the box, which are then all selected in the world. Graphical



(a) Supreme Commander 2 (b) Starcraft 2: Wings of Liberty

Figure 4.15: Selecting a group of units by locality.

feedback of a box showing the area being selected is shown on screen as a 2D element, and the normal graphical feedback for a selected unit (detailed in **unit selection**) within the environment is shown. The objects involved are the selected units, the view camera geometry to determine the world coordinates, and the coordinates themselves. Once the mouse is released (the selection is completed), more feedback and processing takes place. On the details panel, an iconic representation of all of the units selected is shown sorted by unit type, showing the player an easy representation of the makeup of the selected group. Additional processing determines the intersection of actions available to all units selected, and places the set of common actions in the action panel.

Creating a mapping from this interaction to the real world, the input can be transferred explicitly enhancing the sample supervisory interface. Feedback can also be presented in the same way. Processing can be transferred almost exactly, although the camera determination in the 2D map case is much simpler. The requirement for a common set of actions will require a reimagining of the interface in that all robots' actions are not visible at once, and some actions are surfaced beyond the controller. A selection list used instead of a single selection object is required as well. Once these things are added, the processing transfers without issue. A details panel for feedback on group selection can optionally be added as well. This interaction and **unit-specific actions** (discussed later in Section 4.6) have similar underlying model assumptions made about the units being interacted with; a intersection query that is for **unit-specific actions**

performed for every unit in the selected group would suffice for the actions available.

Once a group is selected, along with the limited intersection of unit actions available, the **simple unit movement** and **modal default unit actions** are usually available as well, performing similarly as if each unit was selected in turn and the same interaction performed. This interaction is an excellent example of one which enhances management of multiple robots. It becomes more useful as fan-out increases, as the need to move robots in groups that have heterogeneous sensors or locomotion will be key. It also reduces the total number of actions, as otherwise to move a group of robots, each robot would have to be tasked individually.

Idle Unit Selection

There is an additional selection method available in real-time strategy games. Represented as a button on-screen and available through a keyboard shortcut was **idle unit selection**. In normal gameplay, a set of units are classified as "worker" types, which perform a variety of actions such as constructing and repairing buildings and gathering resources. As resource gathering is competitively advantageous, usually a large number of workers are constructed and tasked across a large area of the map. If any unit is not actively performing a task it can be a disadvantage. It is not uncommon for a unit to be tasked with constructing a building and forgotten. Idle units can be both found and selected by pressing a button, or a key on the keyboard. The implementation of this selection is sometimes limited to the "worker" type unit. Pressing the button multiple times cycles through the idle units.

This interaction is parameterized similar to the **unit selection** interaction, with slightly different inputs, processing, and objects. The input is either the activation of the interface button or keyboard assigned key press. Objects involved are a list of units which are idle, the currently selected unit the view camera object. The processing determines which unit should be selected based on whether the unit currently selected was selected using this method. If there is no current selection or the selected unit was not selected using idle unit selection, then the first unit in the list of idle units is selected. Otherwise, the current selected unit is found on the list and the next unit in the list is selected instead. In either case, the unit is selected with all of the same graphical feedback as **unit selection**, and the camera is moved to center the newly

selected unit on the screen. If no units are idle, the button is greyed out and keyboard key does nothing.

Mapping this interaction starts simply with mapping the view camera to the view port on the robot supervisory interface and the feedback as in the **unit selection** interaction. The processing maintaining a list of idle units must transfer either to the robot middleware determining idle robots. Alternatively a list of idle robots could be filtered from the list of all robots based on movement, or alternatively robots could place themselves on a shared blackboard indicating their idleness after a time without any commands received. The limitation on processing to select only a single type can be discarded. The input can be mapped almost exactly, using a hotkey and button on the GUI.

This interaction can be useful to management of multiple robots where a large number of robots may be available and an operator does not need a specific one for a task. It beings the standard interface dynamically closer to interfaces where task-definition drives the interaction. Normal use of the interface is preserved if a set of robots is desired to be controlled in the traditional way. This reduces the mental load because the decision of which robot to use is eliminated. The status of the button being available or not can also be a signifier that a robot is idle and as a consequence reduce the intervention response time.

Quick Teams

The already outlined selection interactions could be considered the basic methods of selection in a real-time strategy game. Another interaction is common for more advanced players of commander-based games: team assignment and selection. These game interactions make it easy to group units into a team, and reselect that team quickly. Teams are designated by selecting all the units using one of the group selection methods, and then pressing a team hotkey while holding a modifier key. Once the group is designated, a visual feedback appears on the screen to indicate a group has been assigned. The unit in the main view also shows a small number geometrically indicating membership in the team. Once designated, a team can be selected by pressing the team hotkey (without the modifier). The control key is usually the modifier key, and the group assignments called “control groups”. Units can only be part of a single control group. Control groups

remain assigned until no units are left in the group, either by assignment to another team or eliminated from play. The use of control groups is extensive in advanced play, being an easy method to manage a large number of units, without unnecessary cognitive load. These quick teams chunk multiple units into a single mental unit, effectively increasing the fan-out metric of the game.

The **quick teams** behavior is split into two interactions: team assignment and team selection. The input for team assignment is the hotkey activation with the modifier key pressed. Team assignment has a precondition of having a set of units selected, which are involved objects. The mapping from control group keys to units assigned is another object. The processing for team assignment recored the set of selected units in the mapping to the hotkey mapping. Current units in the hotkey group are also removed from the group. Feedback is presented geometrically in the form of a small number next to each unit selected indicating their entry in the team, and an indicator above the details panel appears indicating a group has been assigned.

Mapping this interaction to robotic interfaces is straightforward once **group selection** has been implemented. Input can map directly. Mapping the feedback adds a section to the interface for showing the assigned groups. The only additional objects is the mapping from hotkeys to groups, easily tracked in the interface. The selection interaction is simply another way to implement **group selection**, with the selected units instead of being spatially located, they are selected from the mapping object when the hotkey is pressed.

Action Queue

The last management interactions discussed here is the **action queue**. Action queues were found in both turn-based and real-time commander-based games. Units which have an action queue track a list of actions which are to be performed one after another in a first-in-first-out style. This is useful for actions which take a some time to execute and are repetitive or unlikely to fail, for example building a copies of the same building, or exploring into a uncharted area. The queues holding the actions are limited in length. Usually a queue is appended to by performing the same input sequence used to normally direct the unit, while holding a modifier key. If a unit fails to complete a task that is assigned in their queue, the item is skipped and the next action specified is started.

(a) Supreme Commander 2 (b) Starcraft 2: Wings of Liberty

Figure 4.16: Spatial indicators of queued movement.

Figure 4.17: Queue of build items in Civilization V.

If another action is assigned to a unit, their queue is emptied and replaced with that assignment.

A specialized version of this queue is a continuous action queue. This is most often available or automatically used with resource-gathering units, which gather resources, return them, and then continue gathering until retasked or the resource which is being gathered is depleted. In the real-time strategy games which were examined, an indicator appeared in the game world in the form of a geometric element where queue actions are located, as seen in Figure 4.16. In other games, the queue is shown in a detail panel, such as the one shown in Figure 4.17.

The **action queue** is a complex behavior, broken down into multiple game interactions. Detailed here are the interactions adding an action to the queue, and the automatic progression of the queue. These interactions share some common objects,

including the target unit, and a representation of the queued actions list. When adding an action to the queue, a precondition exists that a unit must be selected. Only actions which relate to the selected unit can be queued. The input to add a queued interaction is the same as when not using the queue, with the addition of a held modifier key. The processing modifies the queue for this unit by adding a record of the next action to be performed to the unit's action queue object. This action queue list object is a modeled as a FIFO queue of objects representing the actions to be performed. Feedback is presented graphically, with a spatial indicator showing in the game world while the unit is selected, and a line connecting the previous queue element's spatial indicator to this element's indicator if the action involves movement. These connected lines produce a line from the selected unit to the location of all the actions within its queue.

The selected unit for this interaction easily maps to a mobile robot selected via a supervisory interface. The mapping for this interaction is most complex when considering the object representing the action specification. This specification can be mapped to a complex middleware system in which each action can be specified in a serializable message, which can then be stored in serialized form in a queue of actions stored either within the middleware or within a queue on the selected robot. The feedback produced on top of the normal action feedback can be mapped directly. Inputs are mapped directly to the original inputs for the interactions queued.

The action queue progresses for a unit through an interaction takes no inputs. Only the unit and its action queue are used. The processing occurs when the unit completes a task, which removes the action at the front of the queue if it exists, and executes the action if possible in the current game state. When mapping this interaction, the queue containing the serialized stored actions represents the queue, and executing the action is performed by deserializing that action and acting as if it was a command just received. The part of processing where a unit finishes its task can be transferred by using a idle detection method as discussed in **idle unit selection**.

The queue of interactions is particularly interesting as it contains other interactions which effectively delays input until a specified time. This reuse of game interactions treating them as commands could produce set of meta-interactions which delay, modify, or compose other more basic interactions. The queue interaction is of specific interest as it reduces the total number of interactive frames when tasking the unit, a desirable

metric to transfer to robotic interfaces. Other game interactions which combine basic interactions can also be explored, although none were discovered in the survey of games here.

4.6 Manipulation

A number of game interactions were found focused on manipulation of the world, usually in the form of opening or manipulating elements to progress in the story. A minor number of interactions involving other objects were also found, and were more prevalent in role-playing and adventure games where the character was more likely to have an inventory of items.

Non-Modal Triggers

Most avatar-based games surveyed had a limited set of actions constantly available using the buttons on a gamepad, or the primary and secondary mouse buttons. These buttons activated the most used actions, including firing the equipped weapon, changing weapons, or reloading. Having this set of actions available at all times, with a large set of actions possible, is valuable for providing easy access to the most common actions which are taken in a game.

The example that is chosen here is a button press to reload the current weapon. The input in this case is the button press. The avatar being controlled is one object, another object being the current ammunition left for the weapon. The processing executes the action associated with the button, in this case replenishing the ammunition in the clip to full and reducing the reserve by an appropriate amount. The feedback for the specific action occurs, which in this case is an animation of the player model.

The mappings for a robotic control system are relatively simple. The processing for the action associated should be replaced with a particular action for the robot which requires no parameters to be specified. In this example, a snapshot from the camera is substituted. The input device can be left the same and button also left unchanged. The robot being controlled represents the avatar. To translate processing, a message can be sent on the middleware to signal a full-resolution photograph should be stored. The feedback animation can be replaced with an iris animation indicating the action

taken.

The **non-modal trigger** is heavily used in most games appearing in half of the games surveyed multiple times. Many in-game actions were activated through a non-modal trigger, either from a keyboard key or buttons on a gamepad. The basic idea of activating an action using a single button press is preserved through the mapping, and is mirrored in some robotic systems today for triggering some automated actions such as grasping an object.

One interesting element of the game interaction that could influence the implementation on a robotic interface is the behavior when the action is invalid at the moment when activated. In the game interaction, **non-modal trigger** feedback when invalid is minimal. This lack of feedback is understood by the player as the action not being available at the time it was attempted. In some games, an explanatory HUD message appeared, in others an auditory feedback was presented. In either case the feedback was unobtrusive and did not distract. This is fine because the cost for attempting to activate an invalid action is small.

Modal Activators

In contrast to the **non-modal trigger**, many games provided for one of a set of context-sensitive actions to be performed by one button press. Pressing the X button might open a door but alternately might pick up an item or manipulate a button, depending on the avatar pose in the game world. Usually the object being manipulated would have a transient HUD display or be highlighted by a **spatial object indicator**. When there is no appropriate object, nothing happens when the button is pressed. A typical example of the **modal activators** interaction is a generic "Use" action. The actual action occurring when the "Use" button is pressed varies: with a door in view, it may open the door. If a computer is in view instead, it can turn it on or off. A switch in view is pressed, some object that is carryable is picked up.

The input for a **modal activator** is a button or keyboard key being pressed. Objects involved are the avatar's pose, the objects within the avatar's field of view, and a list of actions to be performed on specific object types. The processing checks the list of objects within the field of view, starting closest to the camera and moving out, for an object with a valid action in the list. Once an action is found in the list, that action

is performed in the game environment, with associated feedback, usually a diegetic animation or display of some kind. Auditory feedback also plays a sound when the action occurs.

Producing a teleoperation interface translation for this interaction, the avatar pose can be mapped to a controlled robot's camera pose. In the robotic middleware, we can replace the checking of objects in the field of view with a set of object recognizers using computer vision listing the objects in view. Alternately another type of detection could be used, like a proximity reader or fiducial which simplifies detection. The list of actions can then be mapped to a set of default autonomous actions for specific detected object types. The diegetic feedback from the interaction can be ignored as it will happen in the real world in camera. Auditory feedback indicating can be directly copied with a different sound if desired. The input is mapped to a button or key press also without change from the interaction. As a full example, an object detector for detecting charging outlets such as in [54] could be used, and the list could connect that to an autonomous docking and charge action. Another detector included might detect an openable door, and enable the action to push open the door slowly. Both of these actions would be activated using the same button.

Modal activators can take a large set of situational actions not activated simultaneously and ease activation of these actions by compressing activation into a single user interface button. It reduces the complexity of the interface significantly if there are a large number of actions that can be chosen intelligently based on context.

Quick Action Buttons

Present in two of the role-playing games surveyed, as well as one adventure game, was the quick action button. Often used in third-person view avatar-based games, these are buttons which can be assigned by the player to trigger actions to be performed by the avatar. In World of Warcraft[55], up to 12 actions can be selected for the "action bar". This large number of actions available is only a subset of the full set of actions which are available in the game to even a intermediate level player. Some players of World of Warcraft increased the amount of available actions shown on the screen using modifications which were allowed by the games developer[56]. A large number of action buttons makes it possible for the player to have convenient access to many actions at

a time. These quick action buttons can also be assigned to different sets of actions at different points in time. When assigning the actions, an icon representing the action is displayed next to those available. That icon is then used on the action bar to indicate the action assigned. Clicking on the action button executes the action. Some action buttons can also be activated by a keyboard key associated with the button. Diablo III[57] contains two action buttons that can be activated by the mouse buttons.

The **quick action button** behavior is a set of interactions. Identified here are the assignment mode for the action buttons, selection of an action for a particular button, and the activation of a button. World of Warcraft is the source of specific interaction examples.

Input for triggering the assignment mode for the button is pressing the 'P' key, which causing the feedback of a list of all actions available, with their icons, using the interface. There is also a graphical button on screen which can be clicked to trigger this interaction. Objects for this interaction include the list of actions which are available to the player. Each action in the list has associated a unique icon representing that action. The feedback shows the quick action button assignment interface on the screen - this is a large window-like 2D HUD element which has a list of all the actions with their icons nearby. The list contains either pages (with appropriate page-turning buttons) or scrollbars as necessary.

To map these to a teleoperation robotic system, the list of actions which can be performed by the controlled robot is the most problematic. The packaging of a number of autonomous or semi-autonomous actions would be ideal. Integration concerns would most likely cause the list to be retrieved from a robotic middleware system enumerating the autonomous actions the robot is capable of. The list of icons should also be generated using these actions, or automatically assigned as long as unique and distinct. The input key for activating the selection interface can be kept as a key on the keyboard or accessed through the interface using a button to click or another method. The feedback of the interaction should remain the same, with descriptions of the actions which are available for the controlled unit in a list.

The interaction for assigning an action to a quick action button has as prerequisite having the assignment mode already showing its feedback. It takes as input the drag of a mouse starting on an action icon, and ending within a displayed quick action button.

Objects involved are the action associated with the icon being assigned and the list of assignments for the quick action buttons. Processing assigns the action represented by the icon to the position to the correct position in the action button list. Feedback occurs throughout the interaction, with the icon being dragged being duplicated while the interaction is occurring, and replaces the icon (if any) in the quick action button with the icon selected. To producing a mapping for this interaction, most objects can remain unchanged or reused from the previous interaction. A pointing device capable of the dragging action is used for the input, and the previously mentioned list of actions from the middleware or robot is used again to retrieve the action which is being performed and assigned to the list of action buttons. The list of action button assignments is stored internally to the interface.

In activating the quick action button, the input is the mouse cursor clicking on the button or hotkey on the keyboard associated. It has a prerequisite of an action being assigned to that button. In processing for the interaction, a lookup of the action assigned to the button is performed, and the retrieved action is started on the remote robot. The objects in use are the quick action button itself, the list of assignments for the buttons, and the action which being executed. Feedback in the form of a small visual flare or animation on the interface is displayed. In mapping the interaction to a robotic system, the input method and feedback can be kept the same, and the assignment buttons coordinated within the interface as outlined before. The action execution processing can be mapped to a middleware message signaling to start the specific action.

Quick action buttons are a prevalent way for players of games to customize a set of actions available very quickly. Even having a single configurable action button is likely an improvement over a menu interface or two-step action, but they are most often provided in a group to allow customizing a number of actions. This button reduces the amount of time which is spent in interactive state which increases the total task interactivity time. As the autonomy of robots increases, the list of autonomous actions should increase, and the quick action buttons may be an important interaction to translate and test.

Tool Selection

Another game interaction encountered frequently when examining the common avatar-based games is **tool selection**. In most shooter games multiple tools are available but only one can be used at a time. Their mutually exclusivity means the player has some method for switching between tools. Varying effectiveness of these tools for different uses means tool selection happens frequently while playing the games. Two methods were encountered in surveyed games accomplishing this when using a keyboard and mouse. First is to use the mouse scroll wheel, which cycles through the available tools and shows a heads-up display model, as seen in Figure 4.18. When the desired one is highlighted, the player selects it using the primary button on the mouse. The other method for switching tools is pressing the number keys on the top row of the keyboard, which switches directly to the weapon associated with that key. In some games there was also a way to quickly switch back to the previous weapon by pressing a hotkey.

The input to this game interaction is the mouse wheel movement, and the subsequent click to select the tool. Objects involved with the processing include a list of tools available to the player, the tools themselves to be equipped a note of the current tool equipped, a pointer to a highlighted tool and a timeout timer. Feedback occurs throughout the interaction, as processing the mouse wheel movement changes the highlighted weapon, a transparent HUD element is displayed showing the list of tools and the currently highlighted one. When the highlighted item changes, the timeout timer is reset and if it expires the interaction is complete with no other changes. If the mouse is clicked instead, the highlighted tool is equipped to be the current tool. When the

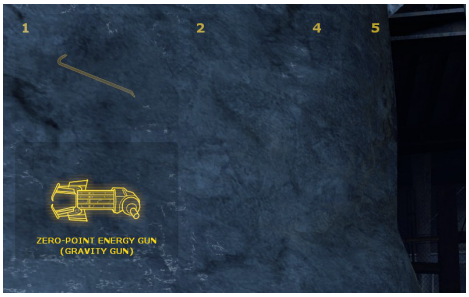


Figure 4.18: Selecting a weapon in Half Life 2[58].

interaction finishes, the HUD display feedback minimizes or disappears. Processing associated with this interaction is the update of the currently highlighted tool, update of the current active tool when the mouse button is clicked, and the maintenance of the timer.

To map these to the teleoperation robot system, a list of tools available must be added to the model. In this example different exclusive modes of a camera are used for the list of tools. The current tool is then analogous to the selected mode. The timeouts, inputs and the feedback can all be preserved through to the robotic interface. When the wheel is moved, icons representing the different modes can be shown with one highlighted indicating which will be switched to when the button is pressed. When selected, the processing switching the tool is replaced by changing modes to the new processing.

This easy selection of modes can be beneficial when one mode is distinctly better for different uses, for example a infrared mode to detect humans versus a traditional camera mode which is better for navigation. Alternately it could be used to switch between two different cameras and modes of those cameras, providing a quick way to assess the situation around the robot. There are a number of different processes within a robotic system which could benefit from a switching tool like this. The example here considers different visual processing modes, but it could map to actual tool changes in the case of a manipulator, where the tool change would happen automatically and the manipulator can return immediately to its last position.

Unit-specific Actions

For supervisory interfaces, players can often choose from a set of actions for a unit to perform which appear in the action panel. Common actions such as move to location, stop, or patrol are available for most units. Many units also enable specialized actions based on their type. A scout themed unit might provide a scanning action temporarily increasing detection range. A jet-pack soldier would not have that action, but instead might have an action to activate jets travel through the air, ignoring obstacles. A transport unit carrying other units would have a specific action for deploying the units at their location. In the **unit-specific actions** interaction, the action panel reconfigures between a large variety of different actions which are provided by a heterogeneous system

when different units are selected.

This interaction has a precondition requiring that a unit be selected. A list of actions available for each unit is one object involved along with the selected unit, and the buttons in the action panel. When the unit is selected, the processing looks up the actions available to the unit and populates the buttons on the action panel with actions represented by icons. This set of buttons is specific to each type of unit so that the correct actions can be shown on the action panel and mapped. A mouse click on one of the unit action buttons, activates the processing of the action which was placed in that button. These actions are usually non-parametric similar to the generic actions discussed when examining the **non-modal actions**, but some require a target. Actions which require a target location or unit will switch the cursor to a mode in which the next position clicked is the target for the action selected. Actions which require a target are not started until the target is specified. Once started, feedback specifically related to the action itself is displayed, along with graphical feedback for the activation of the button, and indicating the change to target mode if necessary for the action.

When mapping to the supervisory robotic interface for this interaction, the action panel feedback which is mentioned in the **unit selection** must be implemented here by placing the buttons. Because of the prerequisite, this interaction requires implementation of some type of unit selection. The list of actions object can be mapped to a list of actions retrieved from a middleware layer which retrieves the services available from a specific robot. Processing placing the buttons in the display can include a rudimentary mapping to icons or short text that can be displayed in the action panel. The rest of mapping is fairly straightforward, with the input click mapped directly along with the feedback as much as possible. The action started from the button lookup is sent as a command to the robot being controlled to begin it's action. Actions which require a target will require an additional processing in the robot interface to correctly represent the target in this message.

With a single action button location taking the place of possibly multiple single use buttons or a menu system, implementation of this interaction can have an effect similar to the **modal activators** by increasing the number of actions which are possible without cluttering the interface. The list of actions in a heterogeneous robotic system would be different for each robot, making this interaction more useful in terms of compression of

action lists when used in such a system.

Modal Default Unit Actions

Another action available when a unit is selected in a commander-based game, specific non-movement actions can be performed using only the secondary mouse button. When the button is clicked on a location which is not empty, a context-dependent action is performed based on the type of the selected unit and the type of the object clicked on. The action taken can involve both objects receiving commands or only the selected unit. For example, if a worker unit is selected, if collectible resources are clicked they will be collected, but if a damaged build is clicked it will be repaired. Military units selected can be sent to attack an enemy unit, or to follow another unit on the team. With a mobile unit selected, and a unit transport targeted, both move to rendezvous at a third location and load the mobile unit onto the carrier.

This interaction is a modification of **simple unit movement** and **modal activators**. A precondition is that a unit is selected. The input is the mouse click. Objects involved include the selected unit and the targeted object under the cursor, as well as a mapping with pairs of types representing selected and targeted types and producing an action to perform. If no action mapping exists, nothing happens. If an action is found, then it is started with the appropriate target and associated feedback. Additional feedback of a blinking circle geometrically located around the targeted object is also included. Path planning occurs for the selected unit and/or the targeted unit in the manner of **simple unit movement**.

To produce a mapping for this interaction into the supervisory interface, the interface outlined in **unit selection** is the base interface. The mapping for the combination of elements into the action should be kept in the interface, and built from the connected robots by querying for actions available through a middleware as described in **unit-specific actions**. Input can be mapped using the secondary button of a pointer still. The processing then looks up the pair in the constructed table and if an action exists, starts it by sending the appropriate messages to the robots in the middleware.

(a) Starcraft 2: Wings of Liberty (b) Portal 2

Figure 4.19: **Multiplayer ping** examples in games.

4.7 Social

Social interaction is a key component in some games, primarily in games with a strong story component with role-playing game elements. Other genres also incorporate a limited amount of social interaction as well. Some massively multi-player games have recently contributed some methods of social interaction which could be considered.

Multiplayer Ping

Three of the multiplayer games reviewed included a separate communication channel which leveraged the spatial nature of the games to provide efficient communication. The communication is implemented in the form of location markers which show on the shared environment of the players on the same team. These are referred to as "pings" by the players of these games, and examples of two different types can be seen in Figure 4.19. This type of communicative interaction is interesting because it represents a communication channel between players apart from the typical voice and text methods. The inclusion of this type of shared state can communicate a high amount of information without requiring significant bandwidth. This type of communication is sometimes limited to a subset of players in first-person team games where it is implemented. It represents a communication similar to a pointed finger in personal body language.

The input to this interaction is a keyboard hotkey or a click of a button on-screen which activates a mode for sending a ping, and then a click on the screen where the ping should appear. The ping location is one object involved, which is transmitted

during the processing to the other players interfaces. The type of ping being sent is also included int he objects. As this is a multiplayer interaction, feedback occurs on more than one screen. First the change in mode is indicated graphically with a change in the cursor local to the initiator. Graphical feedback once placed occurs on all team member's screens, with a marker blinking on the map and spatially in the main view. Additional auditory feedback is played for all players to signal the placement of the marker.

Robotic interactions with multiple operators can use this interaction if mapped for increased coordination. Consider a multi-robot system with two teleoperating robots. The first operator uses their input device to mark a location, by using a button or key to enter the mode and then indicate a location on a shared supervisory control system. The shared location would need to be provided by an established global coordinate system in which case the location can simply be transmitted, or the robot could mark the location by modifying the environment in some way. The alert would then appear on all the other operator's screens, indicating a shared point of interest. The feedback can be directly preserved, or provided in a close approximation.

Multiple-Choice Dialog

Most social interaction in games is very limited, based on the narrative that the game designer provides. In some games, the player is given a method for interrogating non-player characters (NPCs) in the game. This is fairly common in role-playing and adventure games, where dialog with key characters advances the storyline and the player interrogating or revealing certain information can change the reaction of the game. When interacting with a NPC, the dialog for the controlled avatar is usually limited - typically two to five choices are presented for dialog.

The **multiple-choice dialog** usually takes modal input of the controller, where up and down actions of a joystick will choose dialog for the character instead of the default movement interaction. Graphical feedback in the form of a camera move or overlay on the screen indicates to the player that they are within this modal input method, and also presents the choices for the character's dialog, including highlighting the one chosen. A button press will cause the social action to be taken in the form of the character speaking the line of dialog, with it showing either graphically, as auditory dialog, or

both. Objects involved include the player's avatar, the NPC they are interacting with, and a lookup table for the conversation options for the NPC. As the dialog progresses, the processing looks up the current path in the tree, providing the dialog for feedback to the player and the options that are available next. Processing also is used to enter and exit the conversation mode.

When mapping these elements to a robotic interface, the teleoperation interface is used for this example. The input for choosing the dialog can be mapped directly. Processing to enter and exit the modal mode can be mapped to be activated by the player. The NPC can be mapped to a human which has been encountered. The auditory feedback originating from the NPC can be mapped to transferring from a microphone the actual audio. It can also be dropped, although it will reduce the effectiveness. The conversation tree used can be replaced with a menu of options for the robot to respond with, for example asking for an action to be performed such as moving out of the way, or a door to be opened. If context is available, the choices can be picked algorithmically to be quickly useful.

This interaction can be useful for teleoperating remote robots, which can use social interaction to get humans colocated with them to manipulate the world allowing access to more areas for example, or interacting with a victim in a search and rescue operation.

4.8 General observations

Along with the interactions outlined here, games use more traditional methods to present information to and receive input from the player. Most games in the survey included window interfaces or traditional panels of buttons for specific information and data. These interfaces resembled and responded in the "windows, icons, menus, pointer" style exhibited in modern computing interfaces. These window style interfaces were most strongly represented as secondary screens, such as inventory or character status in role-playing games and scoreboard or match setup screens in shooter games. Some of these screens contain interesting interactions (inventory screens seem particularly applicable) but most are operated as with any windowed interface. These interactions can be examined separately but they also should adhere to traditional user interface design rules and guidelines. Examples of good user interface design used in game interfaces are

graying out of disabled elements, use of tool tips to describe parts of the interface after a pointer hesitates, and providing descriptive error messages in exceptional events.

Alongside granular game interactions that can be packaged and transferred to other interface domains, other themes from game design can also be valuable for robotic interfaces. In most games, some input errors are prevented and the immersion into the game is enhanced by capturing and either ignoring or using all input into the system. Each game examined had some type of tutorial or section of the narrative teaching the player how to interact with the game and introducing basic interactions. Most robot operators are taught in a classroom or laboratory setting with one-on-one training with an actual robot. Games as they became more advanced did not have this luxury, and had to teach players how to play the game from scratch. Some of these tutorial methods might be useful in designing a less resource and time-intensive training period for robot operators. Graphics engines which exploit the leading edge in technology have been included in games since the dedicated hardware existed, so the technology used to produce games interfaces should be examined for useful interactions to transfer. Using interface technology developed originally for robotic interfaces can make it easier to transfer discovered game interactions by providing a direct method for reimplementing feedback and input methods.

Input Capture

In all of the games examined, input was restricted to be exclusively interpreted by the game being played. This is the only mode of interaction for console games, there the hardware traditionally had to be reset but now there are special buttons to exiting the game. On all computer games surveyed, the keyboard, mouse and joystick input was completely and exclusively used by the game. The input devices are "captured", useful only for impact on the game. In addition, input from these devices behave differently from standard non-game usage. A keyboard key which would normally trigger one action when pressed, instead performs as a momentary action when held. The modifier keys typically used as momentary might instead be used as a toggle and perform a crouching behavior, or would otherwise modify the behavior of the avatar and have no effect on the other keys' behaviors as in normal applications. In avatar-based games, the pointer is completely divorced from its normal use, and instead is generally used

to orient the camera, providing a rotation around a virtual point using a scheme which has no traditional edge unlike a desktop screen.

This capture of all input increases the player's immersion in the game environment by replacing the actions that are normally taken with the input devices, and replacing them with new actions which control the characters. Capturing input combined with the maximized environment view interaction should increase immersion, possibly increasing situational awareness. It also reduces errors which could be caused by the player - accidentally hitting a keyboard combination which would trigger a non-game action does not cause a break in the action, which could cause a mistake when resuming the game. Some games provide an option reducing this using a "windowed mode". Even in this mode many games retained control of one or both of the input devices.

Capturing input within robot interfaces should be simple, but also produce a similar effect, preventing input errors and increasing situational awareness. Many systems with dedicated control units already perform this interaction simply by the virtue of a specialized system used exclusively for control.

Teaching the player: tutorials

Every video game must solve the problem teaching the player how to interact the game. To appeal to a wide audience, the game should be playable by anyone, even beginners to gaming or players with little experience playing the specific genre. This lack of prerequisite knowledge presents an interesting challenge, in stark contrast to unmanned vehicles, where one-on-one training or classes are more prevalent. As a replacement for traditional training, games present methods for learning the methods of interaction built into the game itself. One possible method to ease the transition to a new game and make it easy for experienced game players is to repeat control trends that appear in previous games of the same type. This can be powerful but there is no guarantee that the player has played games of the same type before, and it can be constricting if new interactions are desired. There is an additional challenge with games, in that a long section of tutorial or training can make a game less fun, resulting in bad reviews or the abandon of a game before it is complete. A balance must be struck making the game fun to learn and keeping the new player in the narrative.

One method that game designers approach this problem is similar to the standardization of WIMP interaction methods – designers and developers have gravitated towards a common set of controls which are used throughout many games. This is seen in the commonality of the control schemes in the **Avatar Movement** interactions across many games. These techniques are prevalent to the degree that when a game breaks from the expected control scheme, it is mentioned in reviews for the game. Game designers balance the concerns of sharing controls with a majority of games with introduction of novel interactions.

Games have arrived at this interface training problem and provide various solutions when starting to play any new game. These training methods are used whether reusing a familiar control scheme or introducing a novel methods. Training in the interactions required to play was usually part of the gameplay experienced in the game survey, as they typically occur near the beginning of gameplay. These training methods were found in found two forms: the tutorial level or in-game hints.

The tutorial level is an area outside the normal narrative gameplay, which is set aside for learning game interactions involved in a game. The goal of this area is to acquaint the player with basic control, introducing usually the movement interactions and showing other interactions in a staged manner, explaining each one either in a building-block style, or independently. When within this area, the set of available tools is often either completely managed or simplified. The task itself, while resembling the tasks to be encountered in the game, was typically of a much lower difficulty than the same task encountered in the game. At the same time, a smaller amount of feedback is present in this area, presumably to highlight the areas where the interface is presenting information to the user. In some games surveyed, tutorial levels had action pause to present an explanation of what is expected of the player. The tutorial level would not continue to progress until the interaction introduced was then practiced by the player.

The training area in some games was loosely integrated into the narrative as a optional task. A non-player character would introduce the player to the interactions required for a basic weapon or technique. Afterwards, the player is allowed to practice with either dummy targets or a non-threatening target to learn the interaction. This technique can sometimes lead to strange dialog from the non-player characters referring to objects beyond the “fourth wall” such as the input devices.

The second most common type of tutorial encountered in the gaming interfaces surveyed was the use of in-game hints. These hints were be presented through HUD overlays, and varied from blindingly obvious to subtle. One example of a subtle hint in a game was a small arrow on the edge of the screen which points toward the next area, or having a goal objects have different shading to indicate it's importance. Other subtle hints might only appear to the player after some demonstration of confusion within the game, such as remaining stationary for too long, or traveling in the wrong direction for more than a specified length of time. Small hints on a minimap or vicinity map might be included. There are also a lot of elements which are built into the game that are much less subtle. In some cases similar to the previous method, the entire game engine pauses the simulation of the world to explain a new interaction to the player, sometimes taking control of the user's actions to explain the methods and show the results. These two extremes of hints represent two ends of a spectrum of interruption for using hints to instruct the player on how to play the game. When using in-game hints to introduce interactions to the player, the right balance must be found to make sure that the player notices the new interactions available but without producing player frustration with the interface.

Interface Technologies

Computer gaming has had an undeniable impact on the graphics technology of computers. Almost every modern computing device has integrated 3D rendering hardware, including recent smartphones. The advancement of dedicated graphics hardware was driven primarily by pressure from the computer customer wanting to play the latest games with high video quality. Graphics cards and chip sets are routinely advertised or sold in conjunction with games. It is rare to see a review of a computer graphics benchmark without a focus on technologies which were introduced through games. Indeed the majority of benchmarks are a scripted set of actions replayed within a specific game's engine. The earliest discrete video cards available to the consumer were marketed and designed specifically to increase performance of the 3D graphics in games.

As graphics acceleration hardware proliferated, the use of the technology in non-gaming applications increased. Many operating systems now use 3D graphics hardware

and techniques to accelerate drawing of 2D screen elements and produce effects enhancing the user's experience. One example of this is Microsoft's Aero interface introduced in Windows Vista which uses graphics hardware to provide translucency and drop shadow effects for 2D windows, and producing limited 3D effects using the same windows. It has also been shown that using a modern user interface compared to interfaces which look old can increase the perception of usability[59].

Given their almost ubiquitous use within gaming intercase, these engines and interfaces should be tested to enhance the operator's experience. In addition, the operators of robotic systems are increasingly more likely to be familiar with game interfaces. Providing control with game interactions in conjunction with a similar interface should provide a more complete experience and lessen frustration. Two types of interaction technologies evolved from gaming are considered: the use of specific interface elements and techniques from modern games to produce a focused and interactive robot control interface, and the use of 3D modeling and rendering technologies to enhance the data displayed to the user. Using accelerated graphics can be applied to even the most basic of robot control interfaces.

4.9 Game Interaction Pitfalls

Even though many of the interactions which have been explored here can be quite beneficial, there are disadvantages to increasing game interactions and introducing these new features to the interface. Commander-style interfaces require a map and localization, which are both uncertain. Avatar-based games present a mobility model which is far more responsive than the typical robot. Robots have a set of sensor which is much larger than most game interfaces are designed to handle.

Commander-style game interfaces introduce a set of requirements and oversimplifications on a robotic system which can be difficult to achieve or justify. These interfaces are based on a global map, requiring a mapping system to be used. In the games, the map is *a priori* known, but most robotic deployments to not contain such a map. Mapping systems used by this type of interface not only need to locate the robots within the environment reliably, but also fuse the map information which is being received

from multiple sources in a way which presents a coherent world view to the user. Advanced multi-robot SLAM methods[51] are desirable for a system such as this, because the robots will need to also know the global position within the environment which is being explored. Even using these methods, the accuracy of such a system in a live environment can be unpredictable. As the game interface has a precise position of all the units available at all times, the localization uncertainty with real-world robots has no analogy in the game interface. A new method to communicate this uncertainty must be developed and introduced to the operator.

The commander-style game interfaces are also optimized for simplified models of units. The units available in the games have very few capabilities, although those capabilities are of a high autonomy level. Most units could be modeled using only a few attributes such as position, health points, and a view radius. Mobile robot systems by contrast are complicated, containing sets of sensors and actuators which can all produce independent information and also have some amount of noise. Displaying these sensors on a commander-style interface presents a challenge for the designer, who must balance the ability to see all of the information with the complexity of the overall interface. Camera-based sensors common in many robots are rare in commander-based games which prefer the world overview and could be difficult to integrate. In this and other ways the bias of commander-style games toward producing and controlling large quantities of relatively simple units work against the robotic interface designer who has only a modest amount of robots to control.

Avatar-based games are also not without their issues when translating to robotic interface. The first-person interfaces in these games are almost always from a front-facing camera controlling a character with a large amount of maneuverability. The relatively slow movement of most robots produces a contrast which introduces delay and would frustrate the operator who is used to the instantaneous response of a game interface. The front-facing bias means that non-front-facing sensors are either ignored or must be displayed in an alternate method, which can require some "mental math" by the operator, increasing the cognitive load. Following on from the commander-based interfaces, the amount of sensors which are available on a typical teleoperated robot is even more likely to be large. Finding a clear method to display all of these sensors can be a daunting task for the interface designer. Game interfaces limit the number of

elements on screen to prevent distracting the player using importance of information as a guide. This may not be viable for a robotic interface that does not know which sensors are important to the specific task it is being used for at the moment.

These issues with game-based interfaces for robotics exist and need to be handled correctly if the game interaction methods which are proposed to improve interfaces are to provide benefit to their fullest. These difficulties must be handled with most robotic interfaces however, and the algorithms which provide solutions in mapping and localization as well as the mobility capabilities of robotic platforms are improving over time. Integrating game interactions should provide increased situational awareness, reduction in training time, and the ability to control robot teams of increasing size. The challenges which are presented to the robotic interface designer should be counterbalanced against the improvements that game interactions can bring to enhance the operator's experience.

4.10 Summary

A set of twenty recent, popular, and well-reviewed games were examined and dissected for game interactions to be produced for the game interaction framework. These games represented a subsection across most genres of video games, representing a variety styles of play and narrative structures. As examined, a dichotomy in the game interaction styles between first-person or avatar-based games and third-person or commander-based games was identified and used to separate interactions which would likely be useful for teleoperation interfaces or supervisory interfaces, respectively.

Common interactions which existed in many games were parameterized using the game interaction framework, identifying the inputs if any, the objects which were involved in the interaction, the processing performed by the game engine to apply the interaction, and the rendering and feedback which was presented back to the player informing them of the changed game world state. After each of these 26 interactions were identified and parameterized, a proposed mapping to integrate the interaction into a generic interface was outlined for each. The ability of the framework to produce these interactions from such a variety of games demonstrates that game interactions can be identified and used from a large subset of all games, and the method should be reusable

for newly released games as well.

Along with the game interactions which were formally identified by the framework, other commonalities between the games that could be beneficial if transferred to robot interfaces were discovered. Methods for teaching the player how to play the game which were integrated into the game experience could reduce the amount of training time required for a operator to completely understand the interface and provide a reduction in costs for training. The games graphics engine technology was identified as another potential gain for integration into the robotic interfaces.

These game interfaces are not without their disadvantages however. They generally use an optimized view of the world, where uncertainty in information availability and accuracy is not handled. They also have issues with latency and the delay introduced by mechanical systems, and the display of the many sensors which are available on the robotic platforms which are normally controlled by the robotic interfaces. These disadvantages should be carefully identified as the assumptions which are made for game interfaces may impact the use of the game interactions when transferred to robotic interfaces. The advantages which are brought by the inclusion of game interactions in the interface should outweigh these disadvantages.

After parameterizing and mapping interactions from games, the mappings should be integrated into interfaces and tested for the efficacy of these interactions in improving the metrics of the interfaces used to control robotic systems. When the framework was under development, multiple interfaces were produced which followed a number of interactions. These interactions were identified and are presented as exemplar interfaces. The interactions which are shared among a number of these interfaces as well as a short summary of each is presented next.

	Game Interaction																										
Game	Translucent HUD	Health Indicator	Activity Log	Minimap	Fog of War	Goal Point Indicator	Spatial object indicators	Quick Action Buttons	Joystick Control	Keybd/Mouse Control	Commander-based Cam Control	Non-Modal Triggers	Modal Activators	Tool Selection	Multiplayer Ping	Activation Time	Cooldown Time	Unit Selection	Unit Actions	Simple Unit Movement	Modal Unit Actions	Group Selection	Idle Unit Selection	Quick Teams	Action Queue	Multiple-Choice Dialog
Mass Effect 3	•	•		•			•	•	•	•		•	•	•			•		•	•						•
Half Life 2: Ep. 2	•	•								•		•	•	•												
Mirror's Edge							•		•	•		•	•													
Left 4 Dead 2	•	•	•				•		•	•		•	•	•												
Starcraft II	•	•	•	•	•		•				•				•			•	•	•	•	•	•	•	•	
Civilization V	•		•	•	•		•				•							•	•	•	•	•	•		•	•
Supreme Com. 2	•	•		•	•		•				•				•			•	•	•	•	•	•	•	•	
Assassin's Creed 4	•	•		•		•	•		•	•		•	•	•			•									
Twilight Princess	•	•		•					•			•														
Walking Dead							•		•				•													•
World of Warcraft	•	•	•	•		•	•	•	•	•						•	•									•
Skyrim	•	•				•	•			•		•	•	•			•									•
Diablo III		•	•	•			•	•			•		•			•	•									•
GTA V	•			•					•	•		•														
Forza 5	•			•						•		•														
Kerbal Space Prog.			•						•				•					•								
The Sims 3		•					•				•							•	•							
Simcity 2013							•				•															
Tropico 4				•		•	•				•															
Portal 2	•								•	•		•			•											

Table 4.5: Interactions discovered in common use in the games reviewed. The **maximize environment view** interaction was used in all games.

Chapter 5

Remote Robotics Interface Studies

When evaluating robot interfaces, comparative studies of multiple options produce valuable results. Typical experiments involving human participants in Human-Robot interactions are not simple to perform. The current typical methods take a lot of time, draw participants from a local area, and as a result have lower numbers of participants. In-person synchronous methods of testing can be identified as a main cause to these issues. Remote self-guided testing has been used in human-computer interaction, and solves many of the same issues while bringing other advantages. With some modifications and augmentation, this type of testing can be applied to robotic interface testing as well.

Many human-robotic interaction studies are performed using typical desktop computer hardware sharing capabilities with most personal computers. These same studies also use simulation software to provide the test environment and sensor data. Using simulation eliminates hardware failures and minimizes time for test setup, making tests quicker to administer, producing more results in the same amount of time. If administered correctly, the simulation, experimental interface and test setup could be executed using most participant's personal computers.

Administrating a study can be automated using a set of techniques to complete the parts of the experiment which is normally completed by the researcher when the study is performed in person. Consent and informational forms normally provided at

the beginning of the study can be presented on a web page. Setup of the simulation can be automated. A record of the run can be created by augmenting the interface with a log of actions performed and data presented. Post-experiment surveys can be presented through the interface as well.

Using these remote study techniques, structured remote robotic user studies introduce a standard method using a client-server model to coordinate the study. It provides a model for administering user studies on new interfaces with remote participants. These studies are easier to administer, and draw from a larger participant pool available through the Internet. It also provides a standardized method for gaining participant's consent, anonymizing data, gathering survey data, and reducing errors in experiment execution. It also monitors submitted data to ensure a correct amount of data for separate experimental treatments are gathered.

The rest of this chapter explores the generation and execution of these structured remote robot studies. First earlier related work in remote usability studies and virtual robotics is considered. The architecture of the server and client parts is presented and all components are described. Three experiments that have been administered while developing the interface are outlined, and study participation reviewed. A comparison of remote studies and traditional in-person methods is provided. Finally some directions for further extensions of the method are proposed.

5.1 Related Work

Remote usability testing has been used sporadically in industry and researched empirically using various methods in the last 15 years. Two types of methods of remote testing have been studied: asynchronous and synchronous. Synchronous methods emphasize simulating a typical laboratory usability test using a video and audio sharing tool, paired often with a remote desktop view tool to record the display on screen. This type of testing was found to be roughly similar to laboratory testing with some advantages including decreased cost and a wider pool of test users. Most difficulties were related to increased setup and technical issues[60].

The focus here is more analogous instead to asynchronous testing, which is less

researched. In comparison studies remote asynchronous usability testing found approximately half of the issues[61] that a synchronous setup had. The advantage in asynchronous testing appeared when the time to prepare and conduct the testing was compared. Even in the worst case, the total time spent conducting the test was 15% of the synchronous testing, and a setup comparable to the type of testing proposed here took only 10% of the total time[62].

An important distinction between remote usability testing in the human-computer interaction field and remote robotics interface testing is the importance of objective metrics and the application of subjective surveys. In the evaluation of robotics interfaces, the objective metrics for judging and the comparative nature encouraged mean that the results may achieve effective results on par with synchronous testing. Similar results can be achieved while gaining the time and increased participant pool advantages from both types of testing.

The use of virtual robotics to carry out these tests is another component of concern, compared to a real or simulated robot locally. Virtual robotics has been used in the recent past to provide methods of training the use of robotic systems without requiring increased monetary investment. Some studies of these training methods comparing real to virtual robots for learning. Tzafestas et al.[63] studied a remote virtual training with telerobotic and virtual with a robot arm, and found that there was no significant difference between the virtual and remote training methods. In this method, instead of training on a virtual robot, the interface is instead being evaluated using a constant virtual robot and scenario to accomplish a specific task.

5.2 Architecture Description

The proposed architecture supporting these remote studies is client-server based, with the client customized to each experiment. Most of the server software is common between all experiments, but some data analysis is custom to each experiment to provide a pleasant experience for the participant and encourage second-hand social media recruitment. The server software supports setup, running, monitoring, and maintenance of the study. Development tools and libraries can be used when producing the client to easily interact with the server, although a standard API is planned. If the server

is customized, logs can be analyzed to provide statistics for the participant and administrators. An outline the typical progression of client-server interaction is shown in Figure 5.1.

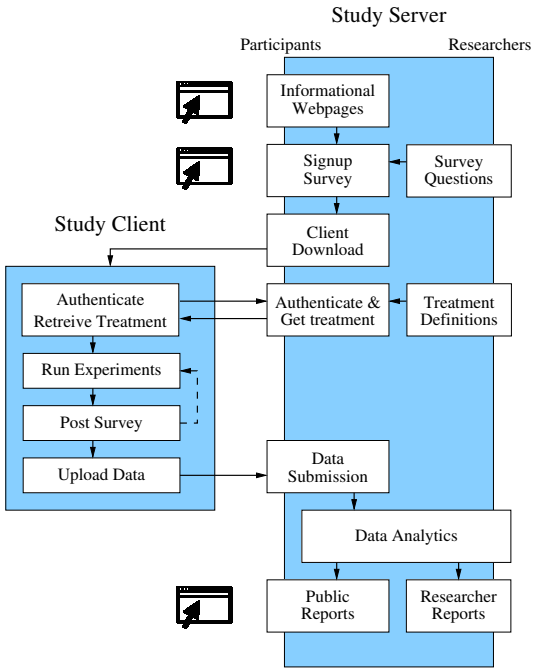


Figure 5.1: Software progression of typical remote study participation run.

Informational webpages are used for advertising and recruitment, and to provide for informed consent of the participants. They are uploaded as free-form HTML and can include images, videos, and other assets. This provides the greatest flexibility. The eligibility questions for each experiment are configurable and can include multiple-choice, check boxes, short answer and essay questions. Any subset of questions can be marked as eligibility questions which are checked against a set of acceptable answers to continue. Other question answers are stored along with the participant's record for later correlation. Questions are ordered, and a subset of the questions defined can be marked as post-experiment survey questions.

Another significant configuration for each study is the definition of treatments and experiment size. Treatment variables are defined with a set number of levels, which also have a value passed to the client for each variable. They can be either within-subject or between-subject. Within-subject variables are passed to the client in an array, to produce a balanced amount of each unique ordering of the variable based on the estimated experiment size. If the experiment size is exceeded, the server attempts to keep different treatments as close as possible.

A final section which can optionally be provided during setup produces analysis of the submitted data. While the data ingress is handled by the server software, the submitted results pages are generic by default, and contain no specific information

beyond confirming that data was successfully stored. Data analysis code can be provided which parses the submitted data, and produces a set of analytics data points for display on the results page. A custom results page if uploaded is provided these analytics in JSON format which can then be displayed dynamically using modern web methods. This provides the most flexibility to the researcher to control the analysis and display of the data to the participant. Analyzed data is separated into two categories: one set available for participants, and another additional set that administrators can access. A separate results page can be provided for display to administrators to view these extra results.

An administration panel is provided where the researchers can log in to monitor the progress of the study, view the researcher-specific results of the data, download the raw submitted data for additional analysis, and exclude submissions that are invalid because of technical errors or client failures. Tracking of the activity related to the study on social media platforms are also provided for convenience.

Study client development primarily consists of the implementation of interfaces to be tested, along with the simulator and test data instrumentation. Two portions of this client communicate with the server. On startup, the server is contacted to confirm the authentication of the study client and retrieve the data on treatments which should be used for this run. Once retrieved, the information is provided to the experiment program to run the experiments while recording any data required. Once the experiment is complete, data is submitted to the server and a URL is provided from the server to complete post-experiment survey answers. If multiple treatments are being tested (within-subjects), these two sections can be repeated with the same or different survey questions. When all testing is done, a URL is provided within the project website for the participant to view their results.

This architecture is flexible enough to support any simulation software library that possible to be run on the participant's computer. The simulation runs separately and different interfaces can be provided in separate executables if necessary. The components provide an interface to the server which is constant throughout different studies.

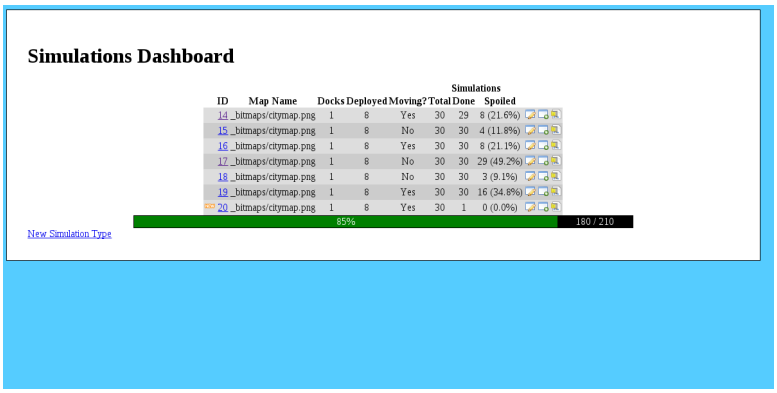
5.3 Experiments

Three different experiments were administered using versions of the remote robotics interface study method while it was developed. The initial software only included the server component, and administered runs of a simulated set of automated experiments for algorithm development. The next experiment included an instrumented client and basic recruiting and participant feedback pages. The latest experiment used the version of the architecture as presented here including a full analysis of each data and social sharing for the participants.

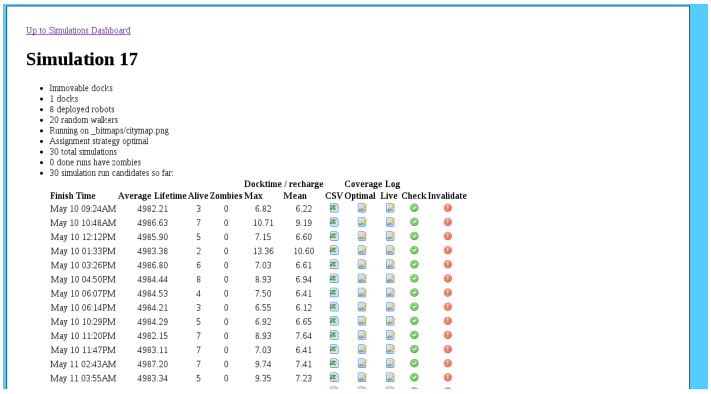
The server software was initially developed to monitor and direct a set of computing resources for simulating multi-robot interaction for marsupial robotics algorithms. The server provided the API for client machines to retrieve their parameters, and the simulation software accepted the simulation results log files and provided analysis of log files to determine when a simulation had failed. The administration interface is pictured in Figure 5.2.

The software was highly effective for monitoring the progress of the simulations and ensuring the correct number of runs for each set of factors was completed, and it was used for two different sets of experiments: one comparing various methods of relocating marsupial docks for maximal longevity[64], and another for relocating mobile cameras for maximal observability of a dynamic scene[65]. 90 runs and 180 valid runs of approximately one hour each were run in each experiment set, with up to 3 varying factors. Runs were spoiled when they were found to contain invalid data either manually or through the rudimentary log analysis provided.

In the second experiment, the remote instrumented interface client was first deployed in conjunction with the server. The experiment focused on selection and teaming methods when moving multiple robots, and is covered in detail with results analysis related to the study in Chapter 8. A recruitment page created and deployed on the server is shown in Figure 5.3. Recruitment for the study was done over approximately 10 days on online forums and by word-of-mouth. 36 data submissions were uploaded to the server over this period. Further analysis of the data provided that one had a technical failure in the instrumented client program, producing at the end 35 valid submitted logs.



(a) Simulation Dashboard



(b) Treatment Summary Page

Figure 5.2: Administration views for remote experiments.

Based on the experiences with the previous experiment, the final remote study expanded on the architecture to perform analytics for the participant immediately after submitting the data. The latest experiment using this framework is the Task Queues study whose specifics and results are covered in detail in Chapter 9. The site was advertised on various social media sites and forums online as well as by word of mouth to gather interest. Within a 26 day window for submissions, 68 submissions were produced. The results were also analyzed by the server and presented in a participant webpage in an effort to increase participation. Each personal results page also contains buttons to share results on various social networks, with the goal to spread the study website and get more participants as well as allow users to share their own statistics. An example results page can be seen in Figure 5.4.

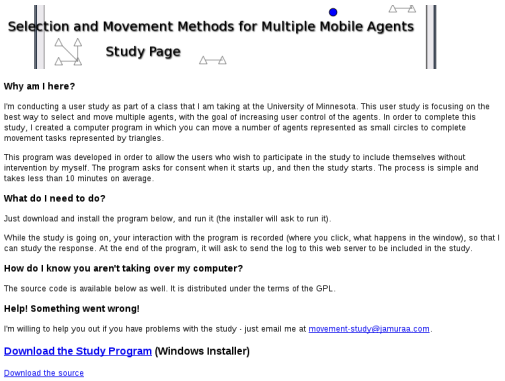


Figure 5.3: Recruitment web site.

These three studies run through the remote study framework provided a way to evaluate how a typical remote robotics interface study can perform. With a minimal amount of effort, participation from a set of users in many different geographical locations that would otherwise not be recruited for a study were contacted and completed the study with minimal resources. This demonstrates the advantages of using the client-server method with instrumented clients.

5.4 Comparison to traditional methods

Using a remote study method such as this instead of the more traditional user studies for robotic interface evaluations introduces a myriad of differences which cannot be separated. Identifying the differences and how they impact the progression of a typical study and gathered data should inform whether a remote study is suitable for a specific experiment. The different stages of experiment design, setup, experiment running and data collection, analysis and follow-up are considered.

For a participant, the experience of participating in the survey is designed to be straightforward and simple. The study's website drives much of the process, including advertising, informational access, on-boarding, initial consent, submitting and sharing of results, secondary social distribution and disclosure of final results the participant if desired. The portion of the study which occurs on the participant's computer is minimized in length, and if the client software is developed correctly, unnecessary leftovers need not be left on a participant's personal computer. This process aims to provide a

Figure 5.4: Example personal results page.

better experience for the participant and provide for the dual goals of welcoming them to the study while also clearly indicating the voluntary nature at every stage.

Compared to the traditional method of participation in a research study, this method presents a number of advantages for the participant. They do not need to travel to the study lab to participate in the study, and do not need to schedule a specific time to complete it. Using the participant's own computer to run the study means that they should be more comfortable in the environment and with the input peripherals used. Anonymous participation is also more available in this method at all stages: the participant receives more information pre-study, during the initial sign-up form, while the study is being conducted, and also are assured they are not discriminated against for non-eligibility reasons. Post-experiment, the immediate feedback on the study website provides a sense of accomplishment and an optional follow-up email informs about the study results in a way typically not provided to participants. The study website also provides an easy contact point for providing feedback and to share information about the study with others.

When designing the experiment, the remote study method can be limiting. Only robot interactions which can be simulated are even possible to include in a framework similar to this. Interactions with other robots is significantly limited by this, and the integration of a robot into a real-world scenarios (interacting with humans). If a specialized controller or specific hardware display is being tested, it will likely be impossible to use a remote study. Even considering these restrictions, most tasks and interactions can be tested in at least a limited manner.

Using the remote study framework, the server can inform based on the configuration of the factors about recommended size of the participant pool for various levels of significance. This can inform the researcher and provide for a reasonably-sized participant pool. After the independent factors are defined, the resulting set of treatments can be reviewed. A limit on the number of participants in each treatment can be provided if some treatments are only desired for minimal pilot studies.

Recruitment of participants can be improved dramatically by multiple factors of remote studies. The availability of the study to everyone who has a suitable computing device increases the potential pool of participants by several orders of magnitude over a traditional study sourcing its participants locally. Beyond the initial pool being larger,

the study participants themselves can spread the word and provide a carry-on effect by encouraging social media sharing. Using the recommended methods, the message on these social media channels can also be suggested. At the same time, filtering participants for eligibility should be marginally easier using the framework.

Diversity of the participant pool should be strengthened using remote studies, as most user studies are traditionally recruited from student populations, which can produce skewed participant pools not only in level of education, but socioeconomic status, gender, locality and other factors. While access to a computer and internet access is still a socioeconomic factor, the pool of participants is theoretically much larger. The previously-mentioned anonymity of the participants can also provide a level of security to a person not available when required to participate in person.

Setup and administration of the experiment requires significantly more work when using remote studies than with a traditional study. The design of web pages to inform about the study is a marginal additional requirement, which might have been done for an in-person user study to provide more information for inquiring people. The other requirement of the remote study is design and implementation of the instrumented client and simulation which executes on consumer-level computing hardware. This instrumentation of interface and simulation is less likely to be used in a production interface.

Designing and implementing this instrumentation through the remote study framework does ensure the same data is recorded for all runs of the interface and produces highly repeatable results. Using a website and program to administer each participation also eliminates minor variations introduced by human error in administration of a traditional study. The even application of the study and the collection of data should provide more stable results. Producing an experiment this way encourages not only reuse of implementation for multiple small variations in experiments, but also validating results by repetition of studies is easier.

The amount of data collectable from the experiment is limited by the method being used in remote studies to what is allowed by the participant's computer, compared to being able to record all parts of the interaction including video of the participant and the interface in a usability lab. Data collectible in the remote situation includes all input that is available to the interface, including all mouse clicks, keyboard interaction,

and other input methods, as well as recording the raw data that is being produced by the simulation system, and the data which is being displayed to the user.

5.5 Future Directions

The remote study framework proposed here can be used to quickly and easily run interface evaluations which are needed to design and evaluate robotic interfaces. Some enhancements can improve some aspects of the remote studies however.

In the immediate future, it is desired to make the server side of the framework available to more researchers to use when implementing their own studies. Currently the framework runs on a distinct server for each different study, meaning that the management of the server must also be undertook by the researchers. An extension to a public server where multiple studies can be run simultaneously would encourage use of the framework for more studies. Along with this, a published API for retrieval of factors for a particular participant and the submission of data will encourage easier development of more clients.

One aspect where in-person testing is superior to the remote asynchronous testing is the monitoring of the participant while the test is being run. Leveraging prevalence of cameras in consumer devices, a recording of the participant could be obtained while using the interface, optionally including prompts to execute a think-aloud protocol. This video would be submitted alongside the data to be reviewed by the researchers as the data is being analyzed.

Simulation fidelity is an area where a higher amount of resources available could increase the realism of the interface usage and bring testing closer to being simulacrum of the real-world robot control. Cloud simulation methods can be implemented where the simulation of the environment and robots is done on a remote server, allowing a more realistic simulation, although possibly increasing latency.

Further advancements to the webpage or recruitment protocols are also considered, as well as providing for the implementation of the robot control interface within the web browser itself using modern HTML techniques and the significantly increased computing flexibility of modern browsers to provide simulation.

5.6 Summary

Remote usability testing for robotics interfaces is a new method proposed here to evaluate differences between different interfaces for robotic systems. This method is completely asynchronous and remote to the researchers, leveraging the participant's computing hardware. By simulating the robots being controlled and instrumenting the interactions with the interface presented to the user, a significant percentage of the data related to interacting with the interface can be collected compared to a traditional user study performed in a lab. This testing has multiple advantages including a larger and more diverse participant pool, easier participation, social network recruitment strategies, consistent administration of the study, and a closing of the feedback loop when the analysis is complete. These advantages are aimed toward producing more results in return for the more complicated setup of the experiment.

Chapter 6

Example Interfaces

In the exploration of the Game Interaction to Robotics framework, games have been examined and sets of common interactions were identified and presented as beneficial for robotic interfaces. To completely use the framework, the final two steps must be exercised as well. Four different interfaces were developed which integrate some of the game interactions already explained along with some new game interactions identified. The final step of the framework must be used to test the interfaces for functionality and their efficacy, efficiency, and subjective metrics used to measure the success of transferring game interactions to a robotic interface. Some game interactions are shared between all interfaces, and each interface also interrogates different and/or additional game interactions. The interfaces' common elements and the interactions from the previous chapter are identified here briefly before being presented in full.

The developed interfaces share some common game interactions. The **maximized environment view** is presented in various forms across all of them Each interface uses a major portion of the interface to view the represented environment or a first person camera view All of the interfaces have a primary input mode for control which does not require moving the hands to perform any of the tasks, allowing the user to place their hands once and operate the interface without moving them. This mirrors most games, where the primary method of input changes very infrequently, if ever, and should provide for an increase in locus of control feeling of the operator.

The first interface, seen in Figure 6.1 is an early stage mobile robotic interface which controls a team of mobile robots under supervisory control. In the interface, robots

are localized using GPS and navigated around an outdoor environment. The system represents robots using an icon, and presents the sensor data spatially on the interface for the operator. **Unit selection**, **simple unit movement**, and an alternate method for **group selection** are implemented in the interface. A succession of experimental trials are used to show the feasibility of the interface for controlling and responding to sensor data on the fly are performed. The interface is used both to identify areas of the environment to avoid, and for setting up waypoints using a path planner. The sensor used is also only providing a small stream of data to the user, but a complex action can be started because of that data. The interface was developed before the framework and was the beginning of identifying methods for incorporating game interactions into robotic interfaces.

The second interface shifts focus onto the control of multiple robots more. An interface is developed to study the differences using various methods of group selection and formation control. It is explored using an instrumented remote evaluation which incorporates simulation and recording of the participant's actions for remote submission. The interface, seen in Figure 6.2, implements **group selection** and **simple unit movement** interactions from the common interactions. Another interaction, **formation movement** is introduced and explored in this interface exclusively. This interface is tested using a client-server remote instrumented evaluation framework which administers a user study without need for the users being in a single location. During the user study, a supervisory interface is used to control a set of simulated robots. The task presented to the participants was a robot coordination task, navigating sets of robots into target formations. This interface also used a tutorial level to introduce the interactions to the participant, and incorporated a qualitative evaluation in the form of a survey. A full description of the remote instrumentation framework, the experiment conducted and conclusions that were inferred from the results of the experiment are presented in Chapter 8.

The last supervisory interface developed shares elements with the first two interfaces. The display of sensor information as spatial information and focus on the global information from the first interface was continued in this interface, which can be viewed in Figure 6.3. **Unit selection**, **simple unit movement**, **fog of war**, and **activation**

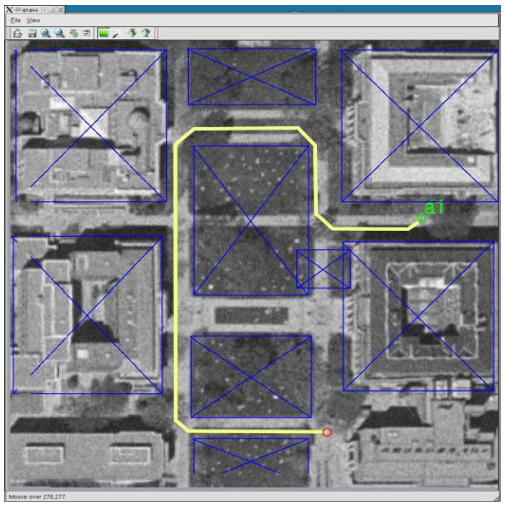


Figure 6.1: SMuRC interface.

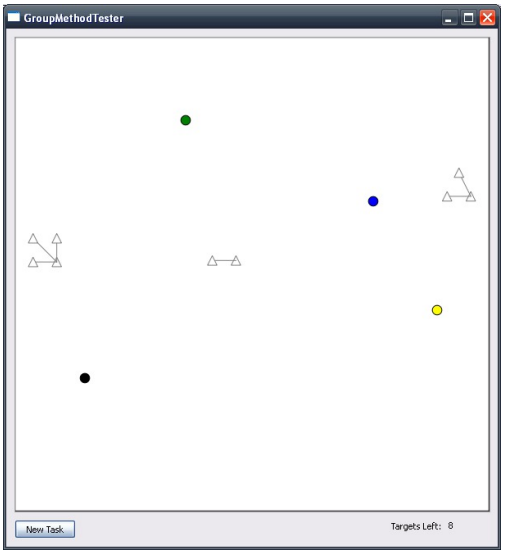


Figure 6.2: Formation study interface.

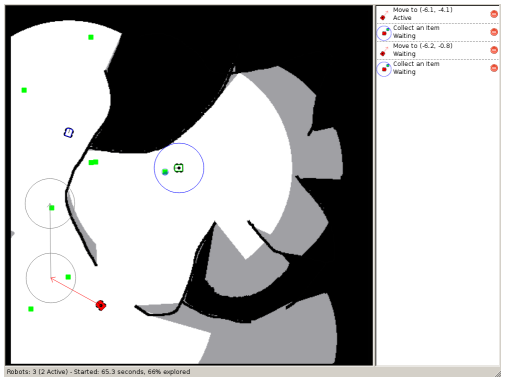


Figure 6.3: Task Queue interface.

Figure 6.4: Augmented reality interface.

time were all implemented and used here in the base interface. The **action queue** interaction is most focused, with the instrumented remote evaluation framework expanded and enhanced for another user study. As in the teaming and selection interface, it includes a tutorial level. This interface also demonstrates how the interactions identified earlier provide multiple robot control with easy changes between the different robots available for use. Subjective evaluation was also continued here, using NASA TLX[22] surveys for participant feedback on the level of cognitive load which was produced by using the various different interactions. The study produced significant data which is analyzed in Chapter 9.

The final interface was purposefully disparate from the other interfaces in multiple ways, showing the versatility of the framework in applying to different situations. The interface uses a joystick instead of a keyboard and mouse for interactions. It also is a single robot teleoperation interface, with a single instead of multiple robots available for control. As a consequence, it uses avatar-based interactions more than the commander-based interactions. The interface developed is shown in Figure 6.4, and is used for a mock search and rescue task focused on exploration. It implements the **joystick movement** game interaction along with the newly introduced **history trail** interaction. The **history trail** interaction is seen through the entire framework in the development and testing of this interface. The interaction enhances exploration by avoiding revisiting areas, and provides artificial landmarks in environments where architectural elements repeat or look very similar. The interface is built using a game graphics engine, leveraging the technology improvements from the intersection of the open source and gaming communities. A usability study is designed and executed assessing the impact of the **history trail** interaction using a variety of objective metrics to measure the efficiency and effectiveness on this task. One advantage which is brought from the framework is that the **history trail** interaction as presented here has minimal requirements on the robot platform being controlled, instead focusing on the feedback and processing needed in the interface. The development and analysis of this teleoperation interface is covered in Chapter 10.

The four interfaces in the next four chapters represent different stages of the development of the Game Interaction to Robotics framework. They have a significant coverage of a useful amount of the interactions of the commander-based interactions which were

shown in the last chapter and represent four concrete examples of the framework transferring interactions from video games into the robotic interfaces which are seen. Within each interface, the last two stages of the framework can be seen. The details of development of each interface is showing that the framework can be used with all varieties of robotic interface: simulated or real-world robots, third-person and first-person interfaces, varying levels of autonomy of robot systems, single or teams of robots, and a variety of sensors and manipulators which are available. The methods for user testing have been outlined here to give an idea of the coverage of the testing methods as well. A range of testing methods have been used in the implementations, from simple experimental trials in the early interfaces to a full usability study in the last interface.

Chapter 7

Simple Mobile Robot Control Interface

Some of the earliest interfaces which were developed for supervisory control of robots were often complicated, or abstract. The goal of the Simple Mobile Robot Control interface was to control a team of mobile robots in real outdoor environments with minimal setup required and a low requirement for obstacle avoidance algorithms on the target robot platforms. The method for achieving this goal was to exploit the capability of humans to identify and mark impassable areas from an aerial photo of an area and perform path planning within the interface.

A secondary goal was to allow the operator to focus more of their attention to the robot and its sensors without multitasking by using spatial interface elements. Spatial elements directly on the map encourage focus on the main working area at all times. In the interface, almost all interaction is done on map itself, except for connecting to the robot and the interface for saving log files, which were placed in dialogs shown when needed. The interface is also meant to respond to user input quickly. The tools available in the Qt toolkit and native language support were used to implement the spatial interface elements and maintain a fast response while communicating with the robots.

The interface was built to communicate with a team of robots that contain GPS sensors for positioning on the map, and was brought through a series of experimental

trials to test the interface and the path planner for viability and reliability. The first scenario involved a movement task, testing the communication and mobility methods included in the interface. The second scenario involved multiple robots, and included two stages, first for dispersal from a common point, and second to rendezvous at a determined point. The final scenario presented was monitoring of a large outdoor area with response by two robots. The operator sent a robot to a predetermined monitoring location, and detect a human intruder using the interface. Based on the direction of travel indicated, the operator would send another robot to one of two areas to investigate.

Interaction with the interface is split into two phases - the setup phase, and the control phase. During the setup phase, an aerial map of the operating area must be provided. A georeferenced and rectified map is preferred to localize using the robot's required GPS sensor. If the map is not georeferenced, three non-collinear known GPS correlations can be used to produce an ad-hoc rectification. After the map setup, the location of the robot is requested to compensate for atmospheric bias in the GPS signal. The final setup task is locating impassable areas on the map. Tools are provided to define rectangular areas. Buildings, water features, steps, and other undesirable areas are identified. After setup is complete, the parameters of the map can be saved to skip steps that are unchanging between runs.

Once the setup stage was complete, the control phase begins. The operator selects and moves the robot or team of robots using the pointing cursor. Sensors are displayed on the map alongside the robots, and each robot can be controlled independently. The **unit selection** interaction activates and selects a robot. Goal locations for navigating the area can then be specified specified using the **simple unit movement** interaction. The path planning incorporated into the **simple unit movement** takes into account the impassable areas defined, with the waypoint planning occurring in the interface. Each robot is expected to also use on-board sensors to avoid local dynamic obstacles.

Experimental trials were performed with the interface to guide iterations of development and determine the ultimate feasibility of the interface for controlling mobile robots. Multiple trials with four different scenarios were completed with four operators who were experienced with the robot. A number of issues were identified with the interface during and after the trials, which were used to guide further development and refinement of the interface. One of the major problems with the trials was identified as

the drift of the GPS system due to atmospheric conditions, which drove inclusion of the robot location gathering step in the setup phase.

The production and testing are revealed in detail, including the interface in Section 7.1, explaining included elements and functions. The robots that the interface controlled and requirements for integrating other robotic platforms are described in Section 7.2. Next, the planning method is explained in Section 7.3. Logs of the trials performed using the interface and robot, including issues encountered, improvements made in response to the issues, and thoughts about GPS-enabled interfaces are presented in Section 7.4. The impact of the interface's development with respect to the Game Interaction to Robot framework is discussed in Section 7.5.

7.1 Detailed Interface Description

The Simple Multi Robot Control interface was developed as a supervisory map-based interface using a georeferenced image as a navigation guide for a group of mobile robots. Previous interfaces for controlling mobile robots in outdoor environments used large databases of images preloaded or fetched from a database requiring storage resources and also using a map corpus which is difficult to update, containing possibly stale information. In contrast, images entered into the SMuRC system can be quickly georeferenced, coming from a more recent source, possibly the same day as the mission when the robot is used. This easy injection of up-to-date data and the simplicity defining the impassable is an innovation which adapts the path planning and interface to a changing environment, making global outdoor navigation much easier than other systems used at the time.

A screen capture of the interface is seen in Figure 6.1. The main element of the interface is the map, occupying more than 90% of the total space. On this map, interactive spatial elements are shown. The interface uses sub-pixel resolution to place the elements on the map to adjust to both low and high resolution images used as maps. Using a higher-density aerial map is preferred as the operator will find it easier to delineate the obstacles and impassable areas. With almost every interaction done spatially, the map element presents a holistic view of the environment with most other interface elements provided as a way to interact with the map or enable elements on

the map. The robots connected to the interface are shown on the map, localized based on their GPS coordinates. When specifying impassable areas, the map is also used to show defined zones and any new area being defined. The interface uses many spatial elements: Connected robots are represented as triangles, the robot location history is shown spatially, selected robots are encircled, and feedback from sensors such as the activity sensor detailed later are shown near the robots parsing the data, as seen with a human activity sensor in Figure 7.5(a). The result of path planning, after it has been computed and is confirmed valid, is shown on the interface as yellow lines through all of the goal waypoints that will be given to the robot, to the goal location.

Multiple robots can communicate with the interface simultaneously. A server running on the robots' middleware presents a minimalistic set of required services: a GPS-based localization reporting, a wireless communication server, and local navigation with obstacle avoidance. SMuRC is intended to be used in outdoor environments where GPS is readily available. The same model GPS was available to use on all three robot models connected. Locomotion is also required to move any robots of course, but in some scenarios one or more stationary devices might also be used. Surveillance cameras and other non-mobile stations could also be connected. An automatic method of connecting is provided by the wireless communication server, or a manual method which can connect any robot even if the broadcast response has been disabled is provided.

Multiple robots can be connected simultaneously, and they can be selected individually or together to send commands to multiple robots at once. To prevent confusion in the interface, a label is assigned to each robot shown spatially near the robot. This identifier is a letter and a number, such as `a1` or `b4`. The designators are used by the operator to distinguish the different robots from each other.

Robots can be selected by clicking on them with the mouse. Selected robots are circumscribed by a dotted circle when selected. A desired goal position is specified using the secondary button on the mouse. The path planning system takes a non-trivial time to compute, so multiple robots can individually be set for movement and once finished, all paths are computed simultaneously. This synchronization causes a simultaneous start of the robots' movement. This behavior is useful to plan out a set of movements for the group before execution. This is a variation from the standard **simple unit movement** interaction. It can be considered an adjustment for a system which

requires a delay when planning. The use of this type of gated coordinated movement allows for the operator to plan formation movements such as rendezvous and deployment as depicted in Figure 7.1.

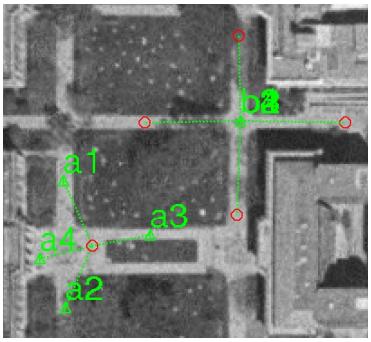


Figure 7.1: Examples of actions involving multiple robots.

Alongside robots, the operator adds obstacles represented by blue boxes with an x drawn within by selecting a button from the toolbar and then specifying the two corner points. As each area is defined, the operator receives feedback - when the first corner is selected, proposed "keep-out" zone is shown as if the second point had been placed under the pointer. The interface was designed to be extended with new elements such as wireless beacons would cause preferred movement, or high traffic areas that should be avoided but are still traversable terrain if required. While the user interface supports adding these alternative elements, they are not used in the experimental trials.

Zooming support was also included in the interface. The sub-pixel positioning accuracy combined with the zoom allows for delicate positioning of elements. The SMuRC interface can be used over larger areas than many interfaces, and could include very large maps. If a large area is being used and a significant number of robots are connected, zooming the interface to an area of interest to complete a particular task can prevent distraction. The zooming interface is designed to support multiple views of the same map in different zoom levels, providing for easy implementation of the **minimap** interaction in the future.

7.2 Hardware Platforms

The interface itself communicates to the robot using the Player[66] protocol. The robot runs a Player server, and the interface is a Player client. The standard interfaces produced through the middleware are the same interfaces and messages that would be used to drive the robot using traditional methods or joystick teleoperation. To use multiple robots with the traditional interface, the operator would switch between the connected robots, or disambiguate three identical interfaces, associating them individually with the correct robot. One advantage of the Player middleware used was the design for integrating multiple robot servers as well as heterogeneous sensors was straightforward.

Three robots were used for the experiments with the SMuRC interface, which are pictured in Figure 7.2. The first robot is a fairly traditional outdoor robot commercially available at the time, an ATRV-Jr. Mounted to the ATRV-Jr platform for communication with the interface and localization were a GPS receiver with antenna and wireless access point. The robot achieves movement using four wheels driven by two motors with skid-steer locomotion. The control software on the ATRV-Jr was limited to in-place turns and forward motion in the experiments, using local path planning based on the GPS localization. The ATRV-Jr was used as the "chase" robot in trial scenarios.

Along with the ATRV-Jr robot, a Pioneer 2 robot was available to the interface. The Pioneer 2 was equipped with the same GPS device model and antenna as the other two robots, but otherwise has a significantly different control architecture for computation and communication than the ATRV-Jr - instead of communicating directly with the motor controllers, the Pioneer 2 has an intermediate control system interpolating

(a) ATRV-Jr. (b) Pioneer 2 (c) MegaScout

Figure 7.2: Robots controlled using the SMuRC interface.

movement commands. It also uses a skid-steer locomotion method, but was less limited in movement and executed turns while in forward motion to complete moving to a specified position. The Pioneer 2 used ultrasonic range detectors to avoid local obstacles while executing the paths provided.

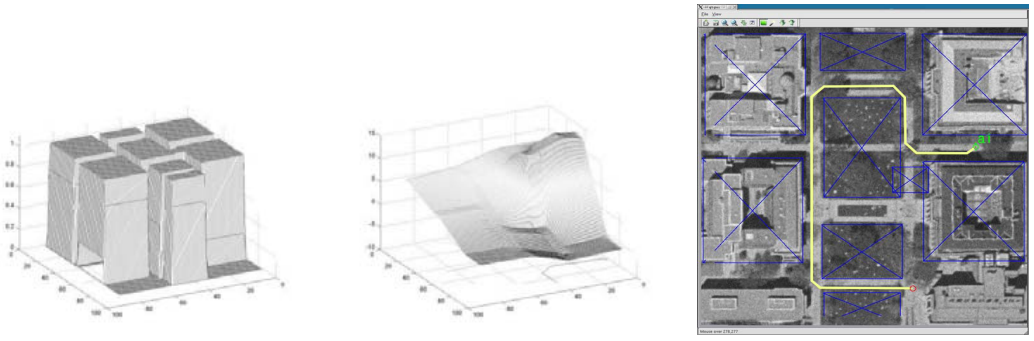
The final platform used in experimentation was the MegaScout[67] which was developed in the Distributed Robotics Lab at the University of Minnesota. The MegaScout is a larger version of the Scout[68] system, a micro robot developed previously at the same lab. The goal for the MegaScout robot was to provide a more capable version of the Scout platform adding a modular bay for sensors and actuators, exploiting the increased size to enable on-board advanced processing of high-bandwidth sensors such as laser range finders and video. The increased volume also provided an opportunity to extend operational runtime with increased battery capacity and traverse more varied terrain with stronger motors and interchangeable wheels. The MegaScout does not contain enough payload space to accommodate the GPS used for the experiments, so it was modified to mount the unit externally. After initial issues receiving GPS data, the antenna was relocated to the tail of the robot for a better orientation. The payload for the MegaScout housed a pyroelectric sensor which can detect at low resolution human body heat and the motion of heat. The pyroelectric sensor readings were classified into three distinct states: motion in a left-to-right trajectory, a right-to-left trajectory, or no movement. The sensor information was relayed back to the SMuRC interface where a spatial arrow indicator of the sensor data is shown next to the robot's icon.

7.3 Path Planning Method

The path planning is an integral part of the interface, providing the major processing required for the **simple unit movement** interaction. The path planning must take into account the defined impassable areas when calculating waypoints from the current robot location to the goal location given. After considering many methods including graph searches (common in computer game development) and artificial potential fields, a method which is based on the electrostatic potential field (EPF) was chosen. An EPF is a variant of artificial potential fields, which apply an attracting force to the goal position and a repelling force from the obstacles and a slight repelling force from the starting

position. This method is flexible for future elements considered for implementation such as relay beacons which could be assigned a slight attractive force to guide the robot near those positions on its way. An overview of the EPF generation and the trajectory generation used in the SMuRC system is presented here; more detailed analysis of the algorithm can be found in [69].

Electrostatic potential fields are built in four steps. Initially, the occupancy map is constructed using the obstacles laid out by the user in the georeferenced overhead map. The occupancy map consists of a square grid which is positioned roughly on line with the pixels of the map provided, but could instead be aligned along a georeferenced orthonormal grid if needed. In the experiments here, all of the imagery was oriented in a north-up rotation and so aligned with the grid. Each cell in the grid is assigned a occupancy metric between 0 and 1 depending on how much it overlaps with the specified obstacles. One generated occupancy map can be seen in Figure 7.3(a).



(a) Occupancy map. (b) Potential map. (c) Final planned path.

Figure 7.3: Stages of path planning.

After the occupancy map is generated, a virtual resistor network is created and filled with conductance values (the inverse of resistance). The resistor network is constructed as a grid with the same size as the occupancy map, with resistors connecting to the eight adjacent cells. This resistor network is then used to generate the potential field by placing a current source at the starting position, and a current sink at the goal position. Applying Kirchhof's Voltage Law ($\sum V = 0$) and Ohm's Law ($V = IR$) over the resulting resistor network yields a system of equations used to solve for the potential field.

This field has desirable properties of the starting position having the highest potential and the final position having the lowest. Equations for the calculation were used from [69]. Figure 7.3 shows the three stages of the generation of the potential field, including the obstacles and the final path generated in Figure 7.3(c) and the potential field generated after applying the voltages in Figure 7.3(b). Once the potential field is generated, global navigation is guaranteed by greedily following the largest drop in potential from the current cell to a neighboring cell. Each cell is navigated by producing a vector from the combination of all the neighbor cell potential drops, which points in the correct direction. The waypoint list produced is then compressed to consist of only the points at the ends of straight segments, which is provided to the robot.

One advantage of using the EPF method is that the paths avoid obstacles, guiding to give ample space in corridors such as the one along the left side of Figure 7.3(c), while balancing it with the total path length as seen where the corners are planned.

7.4 Experimental Trials

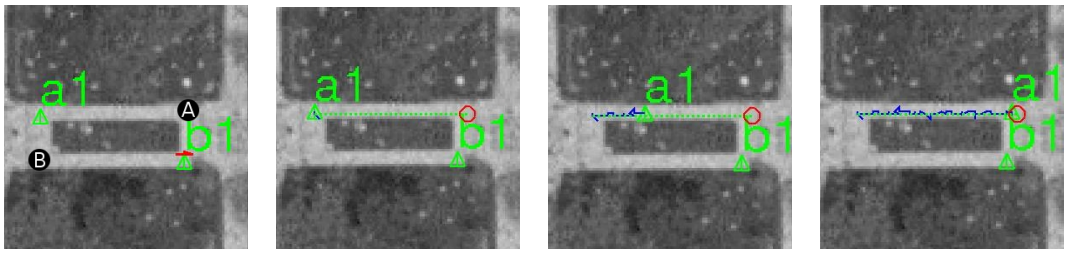
Experimental trials with the system were performed with the interface and robotic systems to determine the suitability of the interface and evaluate the usefulness of the interactions built into the system developed. The first trial tested basic connection and locomotion using the GPS for global localization. Next, a multi-robot trial meant to simulate a detected intruder with one robot detecting the intruder and the other sent to investigate. Finally, a combined trial directing multiple robots was conducted where the robot path planning was used to direct the robots to disperse and then rendezvous. Each trial was repeated at least 3 times.

The ATRV-Jr was connected to the interface and the interface was setup using a georeferenced map with 1 meter per pixel resolution. The robot was reliably able to achieve an adequate GPS fix while setup was occurring. In the trial, a patrol course was selected in which the robot follows a roughly rectangular path around a low-traffic pedestrian area. The desired plotted trajectory shown in yellow on Figure 7.4 represents one run's patrol path followed as close as possible by GPS. In the three runs, the robot followed the path satisfactorily within a margin of error allowed due to the sensor noise from the GPS method used. The history of one of the loops used in the experiment is

shown with the dotted blue line, in which the robot labeled `a1` has traversed the path assigned in a clockwise direction, returning to its starting position.

Figure 7.4: Trial of the system executing a patrol-like path.

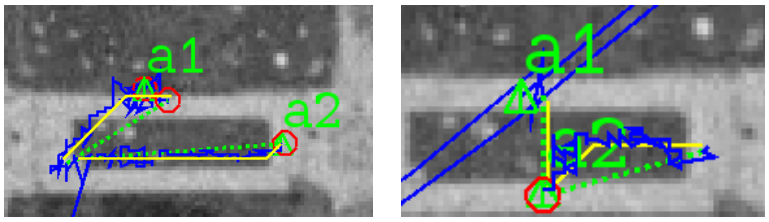
The second trial of the interface demonstrates the ability of the system to easily control multiple robots enabling the operator to quickly assess the overall situation, and select and task robots quickly. In this experiment, the ATRV-Jr was placed along one end of the same pedestrian area, and a MegaScout equipped with a pyroelectric sensor is positioned on the opposite end. The MegaScout then detects a pedestrian crossing the path of the sensor, which causes the interface to display the element for the sensor, showing that a human has traveled in one direction or another. Depending on the direction indicated, the operator tasks the ATRV-Jr to either the indicated spot "A" or "B" to investigate the sensor reading and survey the area. The system automatically plans and sends the necessary path (obscured in Figure 7.5(b) by the source-to-target line), and then completes its movement where it automatically stops. A progression of the interface after the sensor detection in one run is shown in Figure 7.5.



(a) Activity detected. (b) Tasking of robot. (c) Move in progress. (d) Move complete.

Figure 7.5: Trial using multiple robots and sensors.

The third trial of the interface involved simultaneously tasking two robots, a Pioneer 2 and the MegaScout. The two started in the same area and were requested to disperse to different locations, a deployment task shown nearing completion in Figure 7.6(a). The system planned paths for both robots simultaneously and the robots independently navigated to their respective locations. The robots used the GPS localization and local obstacle avoidance to complete their travel through all waypoints provided by the planner. In all runs, the robots reached their destinations, although the different locomotion capabilities caused the robots to finish at different times. In the second part of each run, a rendezvous task was requested in which the robots at separate locations from dispersal were given a common goal point. One issue discovered with the system was over-reliance on GPS for localization and display of the localization information on the display. During one trial run during the rendezvous, the GPS on the ATRV-Jr failed and the robot was stopped. The Pioneer 2 had no failures and completed the assigned course. The interface adapted to the hardware failure and continued tracking the remaining robot without issue. The interface state after this run can be seen on Figure 7.6(b).



(a) Deployment (b) Rendezvous (with GPS failure)

Figure 7.6: Multi-robot trials.

7.5 Summary

The development and implementation of this interface and the inclusion of the game interactions within it present a good example of a low-cost and agile implementation of the Game Interaction to Robotics framework. The use of experimental trials means that the interactions can be tested quickly to determine if they work for their intended goal on a basic level. They are also possible to do without needing a separate base

interface for comparison, which is required of the other methods of interface evaluation.

The Simple Mobile Robot Control Interface was completed and tested using a set of experimental trials. The interface was targeted at controlling mobile robots in an outdoor environment easily and quickly based on aerial imagery and GPS localization. This interface included implementations of **maximized environment view**, **unit selection**, **spatial object indicator**, and **simple unit movement** game interactions. The interactions were integrated together into the interface which connected through a middleware layer to a subset of three robots. Three different scenarios were used to show the feasibility, flexibility, and functionality of the interface. The interface was used in all three scenarios successfully, completing all the tasks that were assigned. The scenarios covered all the different interactions, which were observed to work effectively. The multiple robot selection and command elements implemented in this interface led into the design of the next interface, focused on multiple robot selection and formation movement.

Chapter 8

Simple Teaming Interface Experiment

Multiple robot control and interfaces for selecting and commanding teams of robots were the next target of robotic interface evaluation, after the rudimentary multiple robot control enabled in Chapter 7. As seen from examination of gaming interfaces in Chapter 4 ways to create and manipulate teams quickly are the focus of many commander-based game interactions. An interface was designed to evaluate the usefulness and utility of some of these interactions, including **group selection**. Another **formation movement** interaction which exists alongside these team interactions is examined through the Game Interaction to Robotics framework. The interface used many of the same interface techniques as the preceding SMuRC, including the overhead map basic structure and many of the same interactions. This interface introduced the concept of formation movement which has easily understood rules for team movement allowing the operator to efficiently move multiple robots. The interface also incorporated a tutorial to introduce concepts and lower the learning curve of the interactions. Using an included tutorial and implementing the interface backed by a simulation engine means that in the evaluation of this interface, an instrumented remote evaluation framework using self-guided participation could be used. This framework uses a client-server model to gather result data and track the progress of a user study.

The interface, shown in Figure 6.2 implements a number of game interactions. The

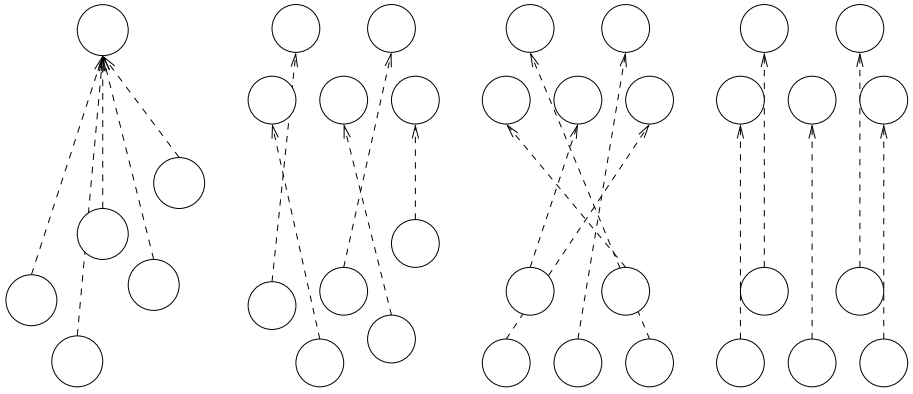
maximized environment view interaction is evidenced by using 86% of the interface as the active area. Almost no space is used on the interface for other controls. The main element in the interface is a map, which is an *a priori* explored binary occupancy map. Additionally as before, all feedback indicating the status of the robots are spatially placed to encourage main area focus. Another feedback inspired from games is used: the display of the current command of an agent when selected. The display of these goal tasks could be considered a variant of the **goal point indicator** interaction. The focal point for evaluation of this interface are the implementations of the **group selection** and new **formation movement** interactions. A user study is designed to discern differences between different implementations of these interactions. The user study was run using the remote instrumentation framework and results are gathered and analyzed.

8.1 Formation Movement

Formation movement is represented by a set of game interactions making it possible to move multiple units with some type of order and minimal interference between the units. This interaction is implemented in multiple real-time strategy games, where a large number of units are controlled at once. Using this type of interaction in a robotic interface should increase fan-out by commanding multiple units simultaneously, which is a key selection heuristic in the framework. The normal movement interaction that which would be issued to each unit if simply duplicated across multiple selected units are modified. If the same command sent using **simple unit movement** were duplicated to all selected units, collisions are highly likely as they traveled to the same point. To avoid this, the game moves the units into a target formation determined by the number and types of selected units. The formation varies, and may be adjustable by the player. The units follow formation rules, with little to no specification from the player. Feedback of spatial elements showing the final positions of the units being moved are transiently shown while the action is being specified. These target locations are calculated based on a number of different factors not communicated to the player.

Most rules used are focused on smooth movement, grouping of similar units, and preventing large movement as a result of small action. One rule prevents a significant number of crossing paths, preventing collisions in the middle of a path. Another common

behavior persists the relative location of the units within the formation, causing units to stay in similar positions and minimizing total movement when the group is moved very slightly. When a heterogeneous set of units are selected and then moved in a formation, they also reorganize into subgroups which are all of a single type. This subgrouping makes it easier to select a single type of unit from the group using spatial selection later. The most sophisticated rules examined included modification of the intermediate movement, with the formation breaking to traverse a narrow path and reforming after there is space available. An example of some of the formation behaviors desired and undesirable behaviors which are prevented by the rules that assign formation movement in game interfaces can be seen Figure 8.1. This is the type of behavior that is being parameterized for the interaction to be replicated in the interface.



(a) Undesirable Clumping (b) Desirable Spread (c) Undesirable Correlation (d) Desirable Correlation

Figure 8.1: Desirable and undesirable formation movement behaviors.

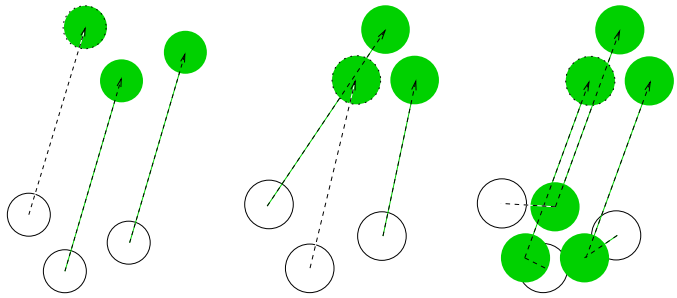
The precondition for the **formation movement** interaction is that multiple units are selected. The input is the same as a **simple unit movement** command, with **formation movement** replacing that interaction when more than one unit is selected. Objects included in the interaction are the set of selected robots and the selected base target position. The processing is a calculation of a set of target positions based on the number and type of units selected, a correspondence between the selected units and each of the target positions, and an optional set of intermediate waypoints for each pair. The target positions and the correspondence are determined by a set of

formation rules. Feedback occurs visually, with the target formation positions being shown spatially near the selected base target position, and path indicators of the units which have movement assigned. Mapping these to the robot interface world, the units are mapped to selected robots in an interface. Base target position can be mapped to any method suitable for specifying a point in the environment, but ideally input using a pointing device activation on an overview map to preserve the interaction. The visual feedback can be brought through directly. The formation positions, correspondence and waypoints by the set of rules, which should be either directly used or reverse-engineered by observation from the game implementation. A subset of these rules can be selected for the implementation.

Two different sets of formation rules were implemented for the interface. In the first set of rules the relative position of each of the robots in the team is preserved in the goal position. First the goal position is compared to the position of each of the robots in the team, and the closest robot is chosen as the anchor robot. The first target position is placed on the goal point, and other positions are placed at positions equal to the difference between the position of that robot and the anchor robot. The correspondence used is one-to-one, with each robot being assigned the position that was created for their position in the original team position. The effect of this is like sliding the entire team as if on a sheet of paper without repositioning them, and is called the *slide* method. When using this set of rules, the operator had to manually place the robots in the formation they desired, either before or after the formation movement, by positioning each robot individually.

The second set of rules used the number of robots selected to choose the target formation locations. The formations used were identical to the target formations required of the interface, shown later in Figure 8.3. The target formation was positioned so the chosen goal point was in its center. The correspondences were chosen from all possible mappings from source to target locations, and the correspondence with the least number of crossing paths was chosen, to prevent collisions. This method worked for the small team sizes used here. This method was called the *destination* method.

A third method for formation movement was also considered but left unimplemented, which used the same target formation as the destination method, but included an intermediate set of waypoints for each path. This intermediate set of points was centered



(a) Slide (b) Destination (c) Intermediate

Figure 8.2: Different methods of formation movement.

in the center of gravity of the selected robots, and was similar to the target formation. Each robot correspondence was chosen in the same way as the destination method, but instead correspondence was targeted to the intermediate method and included minimizing the total distance traveled by the robots when there were no crossings. This method was considered as it is similar to military formations in real-world formation movement. This type of team movement was discarded because the paths are necessarily longer than in the destination method. All three methods for formation movement and assignment are shown in Figure 8.2

8.2 Interface Details

A pointing device is used for all selection and movement. The robots are simulated and represented as colored circles. A robot can be selected using the primary button when it is positioned over the robot. A selected robot is moved by using the secondary button at a target location. This is an implementation of the **simple unit movement** interaction with the path planner disabled, meaning robots attempt to travel directly to the target location given. Simplified **simple unit movement** interactions were used when only one robot was selected.

In combination with the **formation movement** implementation with one of two rule sets enabled, the **group selection** interaction is also implemented to ease unit selection. Two different implementations of the interaction are included, of which only one is enabled at a time. In the first method, the operator selects robots by clicking

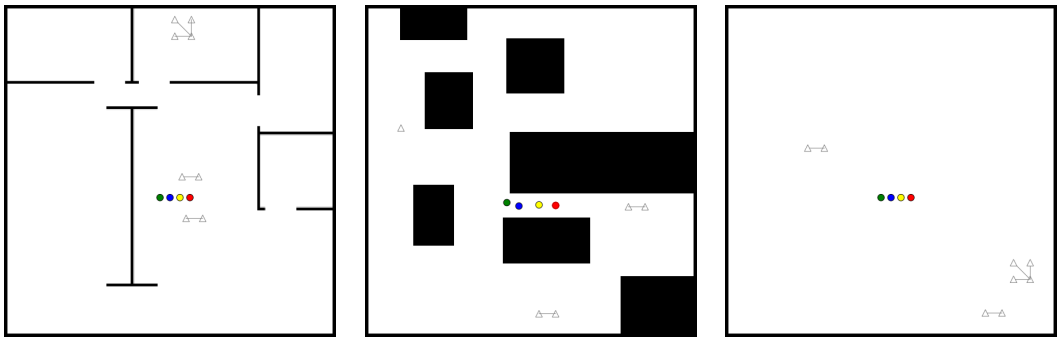
with the primary button. As long as further clicks are positioned on a robot, the robot is added to the selection. Clicking on an empty area with the primary button would cause the selection to be cleared. The second method implemented is the previously explored locality of position method, shown in Figure 4.15. To select multiple units, the operator drags the pointing device from one location to another, with visual feedback showing a box. All robots contained within the box formed are selected when the drag completes.

The robots simulated by the interface are simplified mobile robots in a planar environment. They were identical with holonomic motion and a single movement speed when not stopped. The speed of the robots was chosen to be able to traverse the length of the environment in approximately 6 seconds. They were shown as simple circles which have different colors to distinguish them from each other. Robots that are selected have a red border to the circle, while unselected robots have a black border.

The interface implemented the task to be completed by the participants in the study. This navigation task presents position targets for the robots to fulfill. These position targets have an order between one and four, according to the number of robot locations that are connected and must be filled to achieve the target. Position targets are achieved when a robot is located touching each connected position in the target. Tasks of the same order are always presented in the same constellation, seen in Figure 8.3. To encourage use of the formation movement, the position target constellations and destination formations are identical. Three targets appear simultaneously in the environment, and a new position target is created at a random location when one is achieved until ten targets total are generated.

Three different environments are available for use, shown in Figure 8.4. The first was a simple open area with no obstacles. The second contained obstacles that the operator needs to navigate around to avoid collisions and achieve the tasks. The last is a semi-structured environment with rooms separated by walls with small entrances.

Figure 8.3: Target task examples.



(a) Rooms (b) Blocks (c) Open

Figure 8.4: Environments used, shown with agents and targets.

Some of the entrances are too small for a robot team in formation travel through so the operator using formation movement would have to break and reconstitute the group at some point during room-to-room movement. It was thought this type of environment would discourage use of formation movement.

8.3 Experiment Setup

The experiment was carried out using an early version of the remote robotics interface framework described in Chapter 5. The participant downloaded the instrumented program from the recruitment web site designed and hosted for the movement study. The downloaded executable is then run by a potential participant in the study on their personal computer. The program administered the study protocol, including consent to participate in the study which was presented as soon as the program is started. After consent was received (otherwise the program terminates immediately), the participant was randomly placed by the server in one of four different groups determined by the two independent variables of the study.

The implementation options for the **group selection** and **formation movement** rule sets are the independent variables in the study. **Group selection** is randomly distributed between the "rubber band" and "multiple click" choices. **Formation movement** rule sets are chosen between the slide and destination behaviors. The environment was variable, randomly chosen for each run between the three available.

Questions	Extremes (1 . . . 7)	
How difficult was it to make the agents move where you wanted?	Very Easy	Very Difficult
How difficult was it to select more than one agent?	Very Easy	Very Difficult
How useful was selecting multiple agents instead of one agent?	Very useful	Not useful at all
How often did you select more than one agent at once?	Every time	Not at all
Did you think there were too many tasks on screen at once, or too few?	Too many	Not enough

Table 8.1: Questions posed to participants.

After being placed into their test group, the participant completes a tutorial detailing how to move the robots. The tutorial is comprised of a set of dialogs including interactive areas where the robots are used to advance the tutorial. The tutorial ends in a full-size practice area with sample targets. When the tutorial is complete, the scenario interface for their test group appears and the participant completes the targets generated. After the ten tasks are completed, the experiment is complete. A short survey is administered after the experiment which includes five questions using a seven point scale. Questions focused on subjective difficulty and usefulness of the controls. The questions asked are shown in Table 8.1.

Once the survey is complete, the data was sent to the server which recorded the participation and saves the log files. The application server performs a minimal amount of processing on the submitted data to enable the dashboard-type display summarizing the progress of the study, showing progress in how many total runs submitted and a separate count for each of the four test groups. The data gathered by the instrumentation in the client is extensive. It includes all test group parameters including the selection type, formation rules, and the environment used. It also contains detailed user interaction data, including the position of the pointer when any button was pressed or released, the calculated target position of all agents when a command was sent, and any selections made. Data on the tasks were also included, the position and order of each, and the time when they were generated and completed. Data was gathered for both the tutorial practice area as well as the final active experiment. Finally, the survey results were submitted in the uploaded file. The data gathered were chosen to be sufficient to

replay the interactions submitted in each run. The data gathered is extensive, so an example log is available to the participant before consent is given to reveal the type and amount of data being recorded. An option to review their data before transmitting it to the server for processing is also available.

To judge the different interaction combinations against each other, the first metric tested was total time to task completion. As the tasks were balanced for complexity across the different targets, the total time to complete all 10 tasks was extracted. This objective but also very most coarse and prone to noise. To augment this metric, each task is also considered separately, calculating the time to achieve the task from the completion of the previous task. This efficiency metric can also include noise factors like relative positioning compared to the last task. Travel distance could be partially eliminated by normalizing over distance traveled. The order of the task is also relevant as more resources and coordination are required with increasing task order. As such, higher-order targets are expected to be more difficult to complete and take more time. Total task interactivity time is also approximated by assuming interactive periods beginning on each mouse action with some fixed duration. The total number of interactions is bounded by the total number of mouse actions, counting a dragging mouse selection as a single interaction.

The subjective surveys from the participants are also analyzed in relation to the test groups. The two different types of group selection, and two different rule sets for formation movement were considered separately. Cross-correlation effects were investigated for consideration, but any effects found were small. The surveys proved valuable as some results were found only during analysis of the subjective data.

8.4 Experiment Results

Participation in the study was lackluster; only 35 participants submitted logs into the system, much less than hoped for. The low number of participants meant only 350 target completions were available to consider. Download logs were not available, so the conversion rate from download to completion is unknown. A small number of support emails were received at the project email and technical support was provided, which resulted in successful submissions. The submissions were evenly distributed across all

test groups, as shown in the different group populations in Table 8.2. The amount submitted to each test group was not sufficient to produce significant results directly. The correlations observed are only indications and not significant in many tests. The major problem was the lack of participation. The study was only able to run for limited time, and shortly after was shut down for results to be analyzed. It would have been preferable for the study to continue with more advertising, gathering participants until significant results were achieved.

To analyze the results, first a one-way analysis of variance was performed with the dependent variables of total time to completion and total interaction count, with independent variables being map type, formation rule set, and selection method. Within one variable, the only significant factor was map type, with the empty map being completed on average in half as much time as the other maps ($F(2, 32) = 5.265, p = 0.0106$), and less than half the interactions on average ($F(2, 32) = 7.309, p = 0.0024$). No other significant indicators were found with either one or two factors, likely due to the strong unwanted effects of the map type.

When analyzing the per-task completion times, 340 tasks were completed. The order of the task had a significant effect on the time to complete a task ($F(3, 336) = 14.9, p < 0.001$), with targets of higher order taking more time. The map type was again a significant factor when considering all tasks ($F(2, 337) = 12.68, p < 0.001$), with the empty map understandably making it easier to acquire targets. When eliminating the maps where obstacles are present, the formation method is significant ($F(1, 88) = 7.83, p = 0.006$) with the 'Destination' method being more efficient. The selection does not have any significant effect on efficiency, but the 'Dragging' method encourages multi-unit commands ($F(1, 332) = 5.739, p = 0.017$).

	Selection Type		
Formation	Clicking	Dragging	**Totals**
Null	6	11	17
Destination	8	10	18
Totals	14	21	35

Table 8.2: Test population distribution.

When processing the survey results, the number of position targets available simultaneously was not considered too few or too many ($\bar{x} = 4.1, \sigma = 1.13$) indicating that

the participants were not overwhelmed. Most participants responded that the ability to select multiple agents was useful ($\bar{x} = 1.7, \sigma = 1.51$). Analysis of actual usage shows that around 70% of the time the participant selected more than one robot at a time. The usefulness of multiple selection did not have significant correlation with the formation movement rule set ($F(1, 33) = 1.194, p = 0.282$) as hoped. When comparing different team selection methods, the box selection method was considered easier than the click method ($F(1, 33) = 5.267, p = 0.0282$), a result which correlates with the previously noted increase in multi-unit selection.

		Target Order			
Selection	Formation	1	2	3	4
Click	Destination	21	25	19	15
	Null	12	18	15	15
Box	Destination	21	20	26	23
	Null	26	17	35	32
	Totals	80	80	95	85

Table 8.3: Number of targets generated.

The number of significant results achieved from the user study was poor, and areas of improvement have been identified. The remote instrumentation framework application server and the client performed smoothly for the duration of the user study and were not the cause of any data. The framework was expanded and improved for later interface experiments. The small amount of data gathered in relation to the number of independent variables was the key factor in the sparsity of significant results. When studying the execution of the study, the focus on keeping the length of each run short was likely a detriment to significant data, and the participants could have spent more time with the experiment. The number of factors also should have been reduced by replacing the different maps with a single environment to eliminate the largest irrelevant factor. A pre-study with a smaller number of participants to estimate the effect of these unwanted variables would have likely provided the information needed to simplify the design.

To gather even more data, the between-subjects design could be replaced with a within-subjects design using more than one treatment for each participant. Using a

within-subjects design with two separate treatments per participant would provide correlative data and removed a significant amount of between-subject variation in the data. Within-subjects designs come with other sources of variation including the training effect which requires treatment chains be balanced or the effect estimated and removed.

8.5 Summary

A supervisory control interface employing the **maximized environment view**, **unit selection** and **simple unit movement** interactions to control simulated robots in a planar environment to complete navigation tasks was created as a base robotic interface. On top of this interface, the **group selection** was implemented in two ways, one which corresponded directly with the game interaction it was based on, and an alternate more flexible method. The new game interaction **formation movement** was introduced and parameterized using the Game Interaction to Robotics framework, along with two separate implementations based on differing formation rules. A usability experiment to compare the multiple implementations of these two interactions was designed. A framework for remote instrumented usability studies was constructed using a web server for distributing the simulation client and collect the resulting user study recorded data was used. The simulation client implements the usability study on the participants' computers, self-run by the participants. The results were gathered and analyzed and found that the box method for selection was preferred for multiple robot selection, validating the framework advice to use the game interaction interface methods directly. Other results were marginal but the study as a whole produced a useful real-world test of the remote instrumented study system developed and also provided useful feedback on experiment design for future Game Interaction to Robotics in the next experiments.

Chapter 9

Task Queues

The number of robots which can reliably be controlled simultaneously by the same operator had been studied for more than a decade [20]. A variety of factors affect this metric, commonly referred to as "Fan Out". Autonomy of the robot has been shown to increase this metric [70]. A trained operator can also cause an increase in the number of controllable robots as they learn to more efficiently task single robots. A different interface can improve the Fan Out metric by enabling more natural, efficient or complex interaction with a robot independently of the other factors. The robots controlled can affect the Fan Out in both positive and negative ways - robot systems with more resources and complex algorithms can increase Fan Out, but systems relying on teleoperation generally require more attention for each robot, and decrease the metric instead. Finally, an operator more experienced with a specific system can better predict the future paths and interaction needs of each robot for it to stay on task.

While exploring the current set of games with the Game Interaction to Robotics framework earlier, the **action queue** interaction was proposed to increase the Fan Out metric, interaction efficiency and situational awareness of the operator. Here the interaction is added to a supervisory control base interface to complete the process of the framework. The **action queue** interaction exploits the ability of the operator to plan actions, allowing a user to lay out many steps for a robot. A user study is proposed and executed using an updated version of the remote instrumentation framework introduced earlier. The data gathered is then analyzed to measure the affect of including the **action queue** interaction in the control interface.

9.1 Task Model

The **action queue** interaction requires a set of other actions to exist to be manipulated. It exists as a kind of meta-interaction or simple form of programming which can set into motion other actions based on its simple understandable rules. In this interface the actions a robot performs in response to a sent command are tracked by the interface as **tasks**.

Tasks represent commands which are sent to the robot, and are assumed to require exclusive use of the robot's actuators and sensors. The interface tracks the progression of tasks through three states. In the *waiting* state, the command has not been sent to the robot, most likely because another task is already executing on that robot. In the *active* state, the command has been sent, and has exclusive control of the robot. Once a task relinquishes control it is placed in the *complete* state.

When executing, the active task is referred to as the **current task**, and the other tasks are kept in a first-in-first-out **command queue**. When the current task switches to the *complete* state, it is removed and the next task in the command queue is activated and moved to be the current task. This accomplishes the second, non-interactive part of the **action queue** interaction.

In the interface implemented, all tasks are added to the queue by default, so the basic interactions which command robots are exactly the same when adding the task to the queue. An additional feature allowing the operator to place any task into the *complete* state was added. These *complete* interactions are removed from the queue.

Using this model of task assignment and completion enables an examination of the tasks which are assigned to each robot in a system by the operator, as well as recording the states of the tasks themselves.

Using a command queue of tasks could be considered as a rudimentary method of programming the robot, in fact similar to some simple GUI-based programming methods which have been presented recently [26]. It provides a level of abstraction which is very high level, while still allowing for robots to be controlled individually at a very low level if required.

To better analyze the states that each robot is in over time, we can classify any robot r at time t into an interaction state $IS(r, t)$ as one of:

- Interactive, when tasks are being defined and added to the queue
- Active, when not in the interactive state, but has a current task
- Waiting, when no task is left in the command queue

The natural method of interaction with a robot using this interface is adding a variety of tasks to a robot's queue, then attending to other robots while those tasks run. This causes the interaction state to naturally progress from Interactive to Active and then to the Waiting state, and return to Interactive when the robot is tasked again. Each cycle is termed a **robot interaction frame**, which has a total length from the start of a cycle to the beginning of the next interaction frame. A three-robot system, with robot interaction frames and interaction states notated, is shown in Figure 9.1.

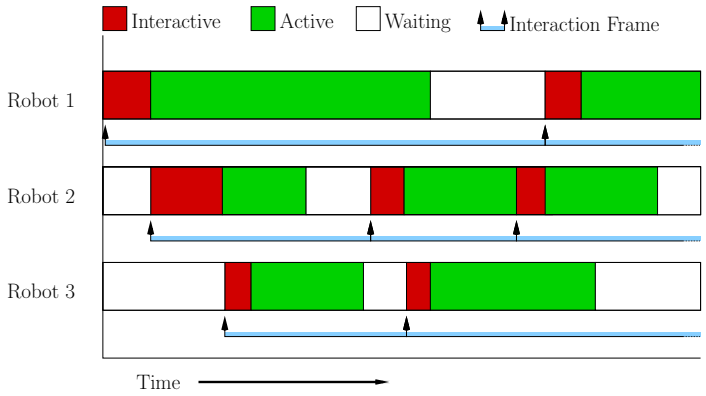


Figure 9.1: Interaction frames of a system with three robots.

This interface enhancement should increase the Fan-Out metric discussed earlier as one of the key metrics. It is not measurable directly in this interface, as neglect time mingles with active time. Two measurable approximations based on the Robot Interaction Frames are used instead. Each of the two metrics can be measured for each action taken, and represent the Fan-Out which could be achieved if all actions were performed the same way. **Theoretical fan-out** $FO_T = AT/IT$ is the theoretical upper limit on the fan-out metric if the robot was attended to as soon as possible. The **actualized fan-out** $FO_A = AT/(IT + WT)$ instead represents the fan-out number which occurred in the real-world system where a robot spent some time in the waiting state, taking a penalty to the Fan-Out due to the operator being distracted or controlling other robots. It is easy to see that $FO_A \leq FO_T$.

When a robot is in the Active or Waiting states, the operator can attend to other robots (which would be in the Interactive state). In the most basic sense, the goal of this research is to decrease the time spent in the Interactive state and increase the amount of time in the Active state. The Waiting state is seen as inefficiency as the robot could possibly be performing useful work, but cannot be addressed as directly as the Interactive or Active states.

As an example, consider a robot with an enabled action queue, supporting *move forward 1 meter* and *turn left 90 degrees* tasks. An operator could add tasks to the queue to move the robot in a square pattern by adding the *move 1m* and *turn 90deg* four times. Each individual task is relatively simple and unlikely to fail so this can combine the total active time of the robot with minimal impact on the interactive time and eliminating any waiting time which would occur between the tasks, increasing the actualized fan-out metrics.

The model chosen here makes an assumption of no communication or optional tasks. This has the advantage that the interaction is simple to understand and use, but not without disadvantages. When a task fails to complete successfully, one of two options exist: either the queue can be stopped and emptied, or the failure can be ignored and the queue continues. If the queue continues on and a navigation task has failed, the remaining tasks remain ignorant of this fact, meaning that some actions are performed in the wrong locations or they may also fail. This implementation compromises by emptying the queue when a navigation error occurs (stalled motor recovery) but otherwise continues with the queue. These disadvantageous situations could instead be mitigated by requiring an operator to be more vigilant to the when it is very important for a task to succeed. In this case, the increased active time of other robots enabled by the command queue could decrease the errors or collisions of the team overall.

9.2 Experiment Setup

An experiment using simulated robots was developed to determine if adding the action queue interaction provides increases either of the beneficial fan-out metrics. The experiment is completed in simulation, so the choice to use the remote instrumentation evaluation framework for a self-guided study was chosen. Part of the reason for using

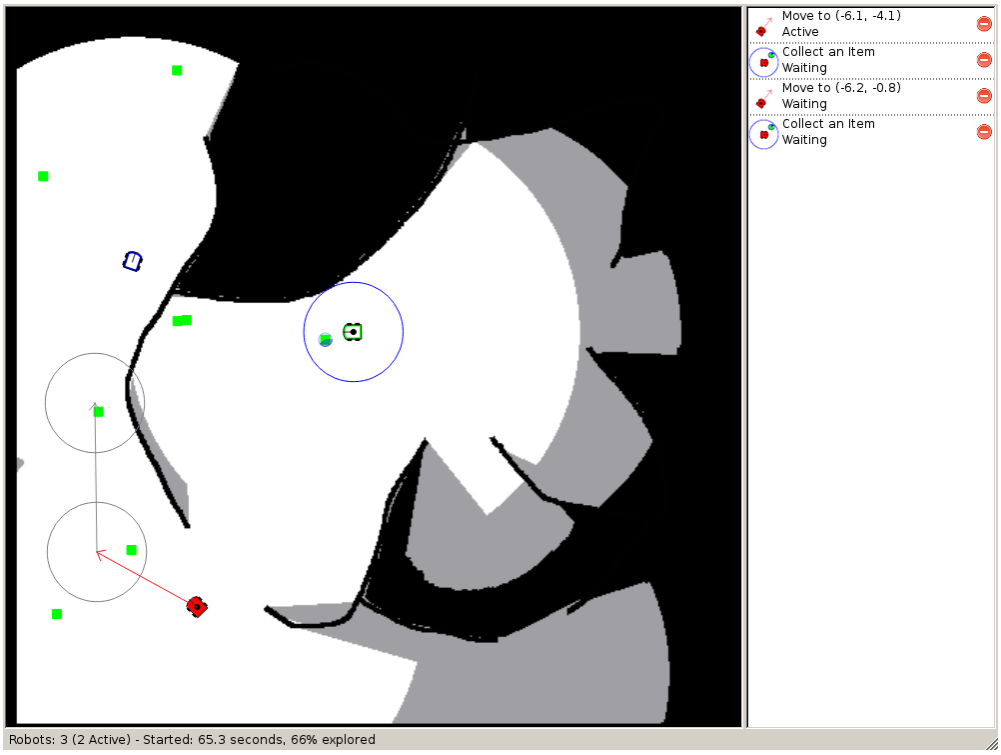


Figure 9.2: Queued actions interface used for experiments.

the framework was to increase potential pool of participants. In the experiment, each participant would again download the study program from the study website, and run the simulation program on their personal computer.

The base interface included enables controlling multiple robots in a planar environment enabling the **action queue** interaction. An example of the basic interface developed is shown in Figure 9.2. The framework client developed can simulate robots using basic internal algorithms, but can also connect to a Player [66] server to control real-world robots or make use of the Stage [71] standalone libraries for a more complex simulation environment.

The target scenario used in the study is an exploration and resource gathering task. Mobile robots are placed in an unexplored structured area which includes a number of target items. The robots can plan around obstacles globally, but not around other robots or dynamic obstacles. The robots construct a map of the area as it is explored and localize themselves within the mapped environment is made. These mapping and

localization technologies are assumed in the internal simulation, and could be accomplished in real-world systems using a choice of localization and mapping technologies available. The scenario is considered complete when all of the target items have been collected and the map of the area is explored at least 95%.

During each scenario, the participant controls three robots. The robots are modeled as skid-steer mobile robots with a omni directional ranging sensor which can detect walls and a target item detector with the same radius. The motion model was chosen because it applies to many types of mobile robots used in research. It is further assumed that mapping identifies walls as distinct from other robots and moving obstacles, excluding the dynamic items from the generated map. The major consequence of this is that revealed sections of the map are persistent and the collectible items are not, requiring the operator to remember their locations if they are not collected.

Each robot can determine if it encounters a motor stall and will cancel the current task and clear all tasks in the queue. A stall recovery task is added which uses alternating motion in an attempt to extricate the robot from a stall automatically. The robots do not use any local obstacle avoidance meaning the operator must avoid having the robots impede each other by monitoring their status. A visual feedback of a warning icon is displayed when a robot is in a stalled state.

Three basic tasks have been implemented which can be added to the queue using the interface, although not all three are enabled in all situations:

Simple Move Performs a direct movement action to the location in global coordinates. No planning or obstacle avoidance is used. It performs a turn to orient the robot towards the goal location and then travels forward until it achieves the goal. This task will switch to a complete state when within a small distance of the target. This task is an implementation of the **simple unit movement** interaction without path planning.

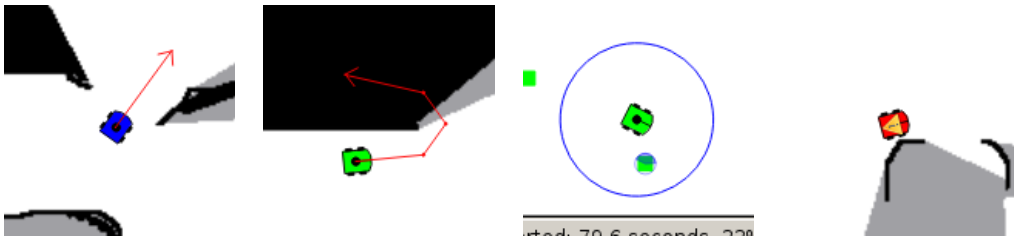
Planned Move Executes a global planning algorithm (currently the wavefront method [72]) and executes the waypoints in the plan in order. Each waypoint is executed similar to a simple move with turn smoothing near waypoints to reduce total path execution time. Periodically, the path is replanned to incorporate new information from the previously generated path. The global planner can fail to generate an

available plan because a path is impossible within the parameters. In this case the **Simple Move** movement method is used instead. The success condition is identical to the Simple Move task - a distance threshold. This task is an implementation of the **simple unit movement** interaction.

Collect Item Identifies if there is a target item within a collection radius (currently double the length of the robot), and attempts to collect it if in range. The task will complete immediately if there are no items inside this radius. If more than one item is within range, the closer item to the robot is collected. The task activates over five seconds, during which no other tasks can be completed. Feedback is shown on the interface as a filling circle overlaying the item being collected. If the timer completes and the item chosen is still present, it is removed from the environment. This task contains an implementation of the **activation time** interaction.

An example of the graphical feedback for each of the three tasks (and the system-initiated stall recovery task) is shown in Figure 9.3.

The interface is designed to maximize the area showing the map of the environment, and minimize the area used for other interface elements, to encourage focus on the environment and robots controlled, another implementation example of the **maximized environment view** interaction. The main elements are an overhead view of the map which occupies a majority of the screen and a display of the current action queue of the selected robot. Robots are represented by small icons which are meant to distinguish different robot types and different robots have unique colors assigned to further distinguish them. Clicking on a robot's icon makes it the selected robot and it is filled



(a) Simple Move (b) Planned Move (c) Collect Item (d) Stall Recovery

Figure 9.3: Spatial elements for tasks.

instead of outlined in the color assigned. In Figure 9.2, the red robot is selected.

The interface also implements the **fog of war** interaction, showing unexplored areas of the current map covered in black. The vision range of each robot unimpeded by the environment within the vision radius is shown uncovered. If an explored area of the environment is not observed it is shown with a gray "fog" overlay. Goal items are not shown when not actively observed.

While there is little technical limitation on the size of the action queue, two robot queue sizes are focused on: the special case in which a robot has only one item available (the current task, a approximation of disabling the action queue interaction) and the case in which the queue size is five. A second variation is a comparison between movement availability, choosing either the Simple Move or the Planned Move task to use for the navigation. Both of these vary so we simplify the choices into two different interaction scenarios.

In the first interaction scenario, the interface used contains the path-planning implementation of the **simple unit movement** interaction, but lacks an **action queue** interaction. This is accomplished by enabling the **Planned Move** and **Collect Item** tasks, and limiting the queue size to a single item. This is the *planned* scenario. The second scenario has only the simplified **simple unit movement** interaction by replacing the **Planned Move** with the **Simple Move** in the enabled tasks, but enables the **action queue** interaction, allowing up to five items to be stored in the command queue. It is labeled the *queued* scenario. The two different scenario parameters are summarized in Table 9.2.

Scenario	Tasks Available	Queue Size
Queued	Simple Move, Collect Item	5
Planned	Planned Move, Collect Item	1

Table 9.1: Scenario parameters.

The task which implements the **simple unit movement** responds to the same input in either scenario: while a robot is selected, clicking on the map with either button will add the movement task available with the goal point at the cursor. A collect item task is added by pressing the space bar on the keyboard. The current task can be stopped and removed from the queue by pressing the "C" key on the keyboard, or by clicking a

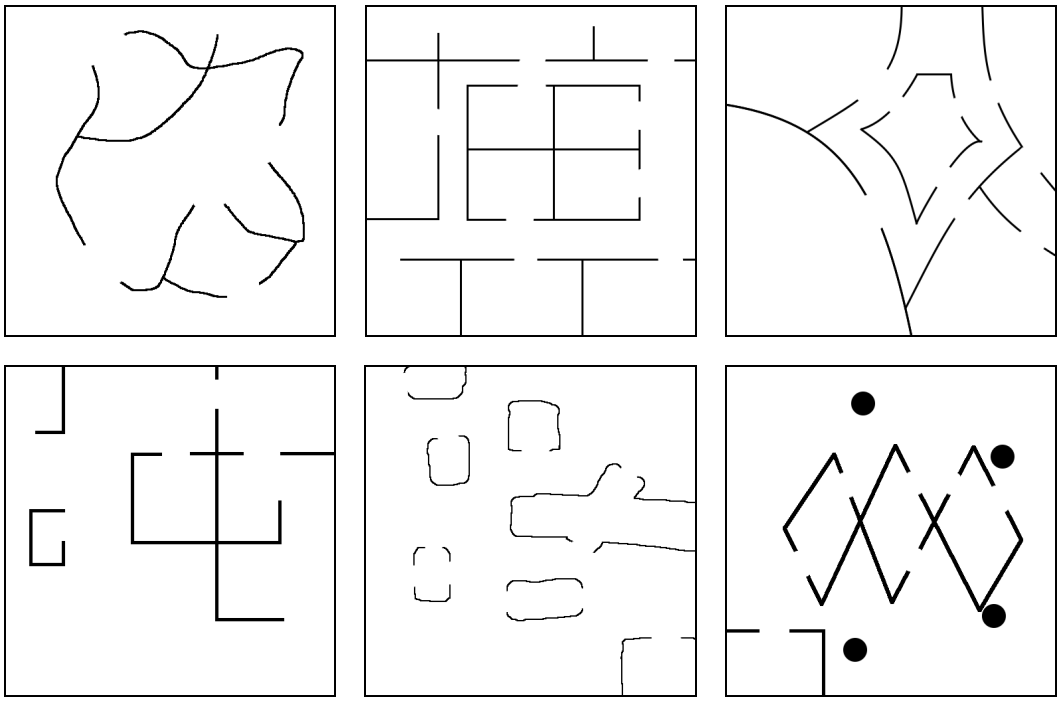


Figure 9.4: Maps used in the experiment.

remove icon next to the task in the queue list on the right.

The experiment is a within-subjects design, with the participant completing comparable tasks using the interface configured for one interaction scenario and then the other. Within each interaction scenario, three maps are completed from the set shown in Figure 9.4. The maps were designed to have similar complexity but still display a level of variation to present unfamiliar environments each time completing the task. A tutorial method is used again here, with the participant completing an interactive tutorial walk through of a typical map. Spatially positioned hint bubbles give the instructions for navigating the robots, calling out specifically the differences between the different methods. The logging was enabled during the tutorial but not used for the analysis phase. Three of the six available maps are randomly chosen to be used in each interaction method, and also randomly chooses which method should be presented first in order to eliminate the training factor. After the third and sixth maps completed, a

modified NASA TLX study is administered to assess the general work load of the interface. The modified TLX study lacks the physical scale from the official scale, which was thought unnecessary. Over the whole experience, each participant in the study completes in order: a tutorial, three maps using one interaction scenario, a TLX task survey, a second tutorial for the other scenario, the three remaining maps using the other interaction scenario and a final TLX survey.

While the program is running, a extensive set of metrics are being logged. At the beginning of each simulation, the map used, interaction scenario chosen and presented, the interaction method and map is recorded. While the scenario runs, the full state of the scenario is recorded at every time step. This includes the percentage of the map explored, the number and position of the remaining collectible items, and a complete robot status for each robot. These robot status reports include position, whether it is selected, the current task executing, and any other tasks held in the command queue. All parameters of each task are also captured including waypoints calculated for the planned move and the time the task was added to the command queue, started, and finished.

Once the end of all scenarios is reached, the program uploads the log to the study web server, which performs a basic analysis of the participant's personal data and presents a page with statistics on how quickly the each map was completed, as well as statistics on the amount of time each robot was spending in each task.

The client program was downloaded from the study website and participants run it on their personal computers. It was designed to run without need to install permanently and without leaving extraneous information on the participant's computer. It was also available at the same time with an identical interface for both the Windows and Mac OS X operating systems. As an incentive to participate in the study, four participants were randomly selected on a set date and presented with a monetary reward in the form of an online gift certificate. The program checks with the server on startup to present the incentive to the users if there is one active. The study design including the integrated program was considered exempt from IRB approval.

9.3 Experiment Results

The experiment was launched on July 6, and the incentive drawing was set to occur August 1, giving a 26 day window for participants to enter the study to be considered for the drawing. It was advertised at various times throughout this period on social media websites to gather interest and participation. The program was downloaded 160 times in this time period. The website received 68 submissions total from these downloads, representing a 42% download to submission rate. Each submission included six scenarios, three each in the Queued and Planned scenarios, for a total of 408 scenarios. The distribution of scenarios to maps was fairly even, with each scenario and map combination receiving an average of 33 entries.

The fan-out metrics of the two different scenarios were measured in three different ways. A direct estimation of the number of robots being used simultaneously is used first, sampling the number of active robots every 5 seconds while each map ran. In most situations, this naive metric can be confounded by factors such as not having enough robots to control, varying levels of workload, or inefficiency in task assignment. The average over an entire map is calculated to get *average robots activated*. When the queue size was larger, the average activated robots showed a significant increase ($t(403) = -8.3, p < .001$) using the Queued interaction scenario as compared to the Planned scenario. The time to complete each scenario also decreased significantly ($t(405) = 2.2, p < 0.05$), with the average dropping from 141.5 to 131.3 seconds.

The queue was used when it was enabled, with an average of 1.94 tasks in the queue at any given time when the robots were active. Robot queues were full approximately 3% of the time, and more than half full 26% of the time. Robots were also more likely to be activated with an item in the queue, which occurred more than half the time.

To measure the theoretical and actualized fan-out represented by the different sets of interactions available, the robot interaction frames of each robot's individual timeline was extracted from the parsed logs. Robots were tracked in the interactive state from the time selected until 2 seconds after the last task was added, or when another robot became interactive. The active classification is given when not interactive but items are still left in the robot's queue. For each robot in each scenario, the number of interaction frames, average frame length, and theoretical and actualized fan-out metrics were found.

Table 9.2 shows a break-down by map and scenario type.

Map	Scenario	n	Frames		Average Fan Out	
			Per Robot	Avg Length	Theoretical	Actualized
boxy	Plan	34	19.43	6.57	3.74	1.55
	Queue	33	10.19	11.23	2.53	1.79
cave	Plan	36	22.05	6.92	4.00	1.72
	Queue	31	12.09	10.31	2.48	1.35
curvy_towne	Plan	32	20.27	7.44	4.04	1.47
	Queue	35	13.47	10.10	2.22	1.39
diamonds	Plan	34	19.32	6.72	3.77	1.33
	Queue	33	12.13	10.80	2.34	1.56
gloop	Plan	38	19.52	6.88	3.70	1.48
	Queue	29	11.71	10.56	2.59	1.61
some_rooms	Plan	27	11.71	11.70	4.15	1.53
	Queue	40	16.57	10.65	2.21	1.34
Total	Plan	201	20.72	7.56	3.89	1.51
	Queue	201	12.85	10.61	2.38	1.50

Table 9.2: Robot Interaction Frame results by map and scenario type.

When analyzing the robot interaction frames extracted from the data, two promising results and a counter-indicating result are indicated. Significantly less frames are used per robot when the queue interaction is enabled ($t(1180) = 23.8, p < .001$). Because a similar amount of work is performed, it is unsurprising that frame length is also significantly longer ($t(13884) = -27.6, p < .001$), 9.9 seconds on average compared to 6.7 for Planned scenarios. Theoretical fan-out is lower with the queue ($t(18837) = 16.5, p < .001$), but there is no significant difference in the Actualized fan-out. This is caused by significantly longer ($t(16140) = 23.1, p < .001$) waiting time ratio ($WT/(AT + WT)$) with an average of almost 1/3rd of frame time spent waiting. By separating the frames produced by different tasks, the cause is revealed as the longer relative active time of the Planned Move action is a relatively high fan-out action, while the Simple Move available on the Queued scenario is less so.

When looking at the subjective task load, the weighted TLX scores indicate that the Queued interaction method was slightly preferred ($t(123) = -2.01, p = 0.04$). This is a promising result given the relatively lower automation required when using the Queue, and the higher complexity of the user interaction associated with the Queue.

9.4 Summary

Queues for actions are used in many computer interfaces today, including download managers, games, and music players; You could also consider analogies in the real world organization of everyday tasks with the simple to do list. A flexible interface was implemented for queued actions on a robot system using a simulator.

An interface was designed integrating game interactions, including the **maximized environment view**, **unit selection**, **simple unit movement**, **activation time**, and **action queue** interactions. To test the effectiveness the **action queue** interaction, it was implemented into a supervisory robot control interface. An user study experiment was designed using this interface through a remote instrumented study framework to collect usability experiment data with the participants controlling a simulated robot team. The framework enabled the collection of data on each participant and uploaded a log for analysis. Analysis of the data received from 68 participants showed that enabling the **action queue** is more effective than using a non-queued method, even when the non-queued method has higher autonomy available.

The *theoretical fan-out* and *actualized fan-out* metrics were also introduced and defined, which are related to earlier Fan-Out metrics but calculable from the available information. These two metrics may be preferred because they are more easily directly measured using simple logs of interaction with a robot interface, instead of requiring approximations of impossible measures of activity or neglect.

Using the Game Interaction to Robotics Framework has improved this supervisory interface showing a viable alternative to adding automation. The ability of the change in interface inspired by the game interactions proved to lower the amount of waiting time significantly. The impact of lowering this waiting time was shown. Using the framework has improved this supervisory interface with little to no modification on the robots being controlled. The level of automation in the interface's queued scenario was low, with almost no advanced planning. The improvements suggested using the Game Interaction to Robotics Framework and the improvements implemented have shown that games can provide useful interactions for supervisory interfaces.

Chapter 10

Augmented Reality Trails

Supervisory interfaces presented in earlier chapters have demonstrated the implementation of a number of interfaces adapted from the game interactions. A majority of games present an avatar-based interface rather than a supervisory interface. A large contingent robots in the field are operated primarily by teleoperation, including the majority of urban search and rescue robots, vehicle inspection, bomb disposal and reconnaissance robots. For these "dangerous and/or dirty" tasks, the current best method to complete the required tasks is teleoperation. The high bandwidth of information presented through a direct video feed combined the ability of the human operator to make decisions and identify targets in real-time makes teleoperation the preferred solution. The operator can process the information that is displayed by the video feed when it is often difficult or impossible for it to be completed in an automated manner. Teleoperation can be used in situations where it would be impossible to use another type of interface, and requires less sophisticated sensor equipment. All of the supervisory interfaces presented earlier require a mapping technology to operate on the basic level, along with accurate localization within the environment. Teleoperation requires nothing beyond a video feed and some control over locomotion. While other interfaces may confine movement to an area where the robots can be tracked or can be displayed on a map, teleoperated robots can potentially reach any area that they are mechanically capable of reaching. The Game Interface to Robotics framework can be used to improve and enhance teleoperation interfaces, using the same methods which have been demonstrated for supervisory interfaces. Demonstrated here, an teleoperation interface is implemented demonstrating

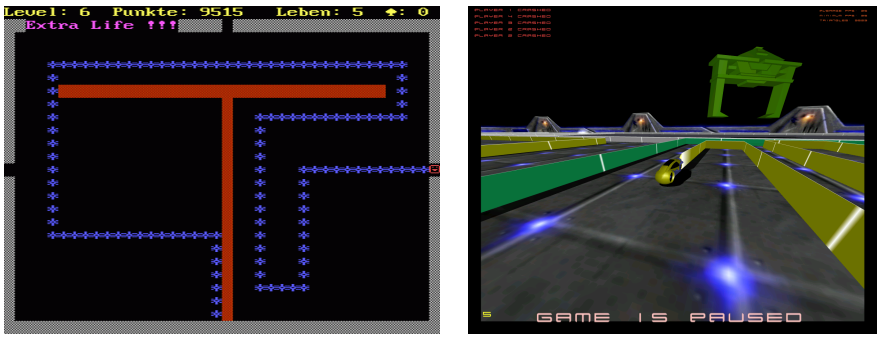
the different target interaction paradigms used when incorporating game interactions.

Using the framework to target these paradigms, the graphical elements from a first-person shooter interface are used as the starting point. First person shooter and teleoperating robot interfaces share a number of similar interactions. In both, the user is shown video images while physically disconnected from the interface being used to control. Small movements in the control surfaces or adjustments can cause a large feedback, either in the view shown or the actions of the controlled object. Both also present a view which represents a restricted field of vision from a forward-facing perspective, even though it is technically possible in most situations to produce a view with a larger angle. Views could theoretically present a panoramic view around the avatar or robot. Typically a view which is representative of the human field of vision is presented. This is to increase the effect of telepresence desired by the creator of the game and the interface used for teleoperation. In both teleoperative robotics and FPS game interfaces, this immersion effect is desirable as it increases the operator's focus as well as the situational awareness overall. In games, this effect is used for the escapism desirable from a successful game, and in robotics, it is desirable to more effectively operate the robot through situations by increasing situational awareness and attentiveness.

The interface that is presented here implements interactions from the FPS interfaces examined in Chapter 4 and implements a new interface which is tested using a mock search and rescue task. In the interface, both larger overarching interface interactions like **maximized environment view** and **input capture** eliminate distractions and increase the telepresence effect as well as smaller modular interactions such as using **translucent HUD element**s for transient information displays and a reticle on the view which enhances the control by presenting a clear view of where the avatar or robot will travel are implemented. A control method is developed sharing similarities to the **joystick movement**, using the joystick input to control velocity and turning. The inclusion of these interactions from first-person games is meant to provide a level of interaction which is familiar to new robot operators who have played these games before, while providing controls easy to learn and use for any operator. The design of these interfaces using these game interactions provides one element this familiarity.

10.1 History Trail Interaction

In addition to the interactions which are considered part of the baseline interface, an additional game interaction was identified which is driven through the Game Interactions to Robotics framework and implemented into a real world interface: the **history trail**. History trails can be found in many different games, but the inspiration for this interaction comes from the classic Snake game and more recent GLtron, pictured in Figure 10.1. In both of these games, an interaction occurs where a trail is presented behind the movement of the player. There are no inputs for this interaction. The feedback in both cases is a trail which is presented behind the player's avatar as movement occurs. The objects involved interaction are the player's avatar and the trail, which can be parameterized as a single or multiple objects. The processing in the interaction includes the production of the trail object or objects, recording the history where the avatar has moved from. When mapping these to a teleoperation robot interface, the avatar is mapped to a robot being controlled. The trail presents a challenge, because it is being generated by the game rules dynamically. Methods which leave a physical trail of crumbs behind a robot can considered, but are undesirable because they modify the environment, a complicated act to perform reliably, and therefore have a limited use. Instead the feedback on the screen is reproduced using rendered elements overlaid on the screen.



(a) Snake (b) GLtron

Figure 10.1: Games exhibiting the **history trail** interaction.

The feedback being added to the screen is a historical indicator of previous locations that the robot traversed. Envisioned as a trail that shows as a ribbon or pipe in

the world view presented to the operator, translucent and visible whenever the operator teleoperation view shows an area which was already traversed by the robot. This display of the history of the robot should enhance the sense of place, highlighting the locations which have already been visited by the operator, information which provides a variety of benefits. Firstly, the operator can use the history trail in a "breadcrumb" approach to return along the same path they had entered by retracing their steps to the entrance or starting point. Confusing environments or repetitive architecture are also disambiguated, as the trail marks areas of the environment already explored to make exploration more efficient in situations where a whole area must be searched. When exploring an areas which contains a looped path, this loop in the environment is more easily noticed with the trail showing, preventing situations where circular paths confuse and disorient the operator. It could also be advantageous to see the trail "through walls" to orient the operator in relation to the location that they entered the environment within if they can see the origin.

The trail is implemented using an augmented reality method to display a virtual trail on top of the real world interface which is presented through the video view. The visualization is added to the operator interface, making it applicable to a wide variety of robots and sensors. It requires no additional software on the robot which is being operated, and has minimal requirements for sensing. The only requirement is that robot must supply only some method of odometry. The system can be implemented even using dead reckoning systems, although it will exhibit drift of path odometry data which is common in uncorrected odometry, making the historical path less useful over time as drift occurs. The system will work with any method of odometry, meaning more accurate localization methods incorporating error-correction can be used without any change to the interface, the algorithm will take advantage of the more accurate trail.

One disadvantage of the interface enhancement, the historical data path slightly occludes the original view of the robotic platform, meaning that some details might be missed by the operator if it is not possible to turn the display off so areas can be inspected without occlusion. The translucency of the path partially mitigates this effect, but to be effective as a visible history path, some occlusion of the environment is necessary. The addition of the path as an overlay on the view also requires a significant amount of processing on the device rendering the data. The architecture also must add the path

on the video before displaying the video to the operator of the robot. This additional processing can add latency depending on the graphics capabilities of the viewing device, with higher latencies causing muddy and/or lagged control response.

To test the impact and effect of the **history trail** interaction, and gather feedback for the new display inspired by the gaming interaction, an interface was developed using these elements and a user study was performed where the historical path display was evaluated with regards to a simulated search and rescue task.

10.2 Teleoperation and Wayfinding

The main problem the interaction hopes to address is the wayfinding problem, which occurs when in robotics teleoperation frequently as part of navigation. Wayfinding problems occur when someone wants to get from where they are to a target location, and must find a path in an unfamiliar environment. It is a problem encountered frequently in every day life. Wayfinding is a common enough problem that research on it spans a number of fields, including architecture, where the design of buildings and building signage is modified to make it easier for wayfinding, and city planning, to guide people around and between different sections of a city. Finding the way to your professor's office for office hours or to the food court in the mall, are both wayfinding activities. There are also anti-patterns in this area, such as hiding exits and repeating structures in casinos or fun house mazes.

The nature of search and rescue robotics means that they are often used in scenarios and environments where the operator has little knowledge of the structure beforehand. Wayfinding is a pervasive subtask within almost every task in search and rescue, including exploration (find uncovered areas), Item retrieval (find a path to the item and back) and rescue scenarios (guide the rescuers to discovered items, return the robot safely to the recovery point). This wayfinding problem is often left for the operator to handle on their own, but sometimes handled in one of three ways. The blueprints of the building being explored are retrieved and used by the operator of the teleoperated robot as a map. This is not ideal because changes in the building may post-date the blueprints can invalidate the usefulness. If the incident that caused the need for the rescue was significant, the structure of the building could be compromised, creating obstacles not

displayed on a blueprint. Another way is to use the existing wayfinding assistance made available in modern building design. Obviously this doesn't work if the structure is old or disused, and even if the structure is new enough to include modern wayfinding techniques, the same issues of compromised building structure also apply to the signs and architectural cues found in the building. Even in ideal cases, the disconnected nature of teleoperation can be mentally taxing on the operator and disorientation is common in modern buildings with many repeating structures.

The last method commonly used in teleoperation exploration missions is a generated map. There are two ways a map is generated - either ad-hoc by the operator or assistant, or automatically by the robotic architecture that is being used. Often mobile robot systems provide Simultaneous Localization and Mapping (SLAM) methods to create a map "on-the-fly". The maps often focus on the third-person perspective, displaying the map from a top-down or "over-the-shoulder" view. Using a third-person map requires a mental context switch for the user, even when the maps are oriented such that the robot's front is always facing in the up direction. Systems that provide SLAM are often much more complex as they require accurate sensors, which can be high bandwidth or expensive. In rescue scenarios when the robotic equipment may be harmed or lost, cost of a lost sensor may mean it cannot be replaced quickly or at all. When an assistant is helping provide the map for the structure, inaccuracies can be easily introduced to the map, and it requires additional staff for updating and relaying commands to the operator.

In this system, the previous path of the robot is overlaid directly in the same view as the operator is normally viewing, as if the operator had been dropping a string on the ground when driving, or similar to using a tether in a human-powered operation to ensure the return. The display being in the same view the user is already focusing on means that they have less context switching and mental math and can focus on the task. The minimal nature of the previous path being displayed as part of the environment means that it is unobtrusive, allowing the structure of that environment to take the primary focus of the user. This should provide for the task to be completed more quickly, especially in cases where there is confusing architecture, loops, or repeating structures, and trivializing the common wayfinding task of returning to the entrance of an area.

10.3 Related Work

A surfeit of related work exists in psychology, human factors, and architecture on wayfinding. One specifically of interest in wayfinding relates the availability of a map to the subject in a wayfinding activity and whether it is a help or hinder to them. Goldiez[73] examines the availability and structure of map interfaces for wayfinding with a human navigating a maze which is structured and has a dual goal. They used a constructed maze to determine the effectiveness of different types of maps available to the participants on a task which required them to find a goals in a maze. The results showed that those who had the map available to them when they were within the maze did not have better performance compared to the map being available outside the maze. A control group which did not have a map at all also was marginally better than having a map within the maze. This type of wayfinding using a map is similar, but this work is focused on the generation of a history - none of the approaches from Goldiez included a historical path or a way to modify the map given. The methodology of the wayfinding task is somewhat similar to the task used here, but the one used is modified to be significantly more complex.

Swarm systems based on biological pheromone trails analogous to the trails that ants use in their path finding methods are related as well, even those systems typically focus on discovery and emergent behaviors instead of wayfinding as a general topic. Mayet et al.[74] created behaviors using small swarm robots, which were placed in an environment where phosphorescent paint can be activated by the robots to signal to other robots in the swarm.

10.3.1 Augmented Reality

Augmented reality (AR) is a method where objects which are not part of the real world are inserted into a person's view, providing an environment which is made up of the real world which is captured from a camera and modifies it in some way. This can be used to insert virtual objects, obscure objects that would normally be visible, or modify the appearance of existing objects. A large number of augmented reality applications exist, ranging from medical procedures where images are overlaid on surgery patients[75] to enhancing social interaction by providing extra information about a conversation

partner[76]. Krevelen and Poelman[77] provide a good overview of current technology used for augmented reality and a survey of applications.

It is worth noting that many augmented reality applications developed are games or game-like, with virtual opponents, or virtual actions which are taken against other players of the game. These alternate reality games can range from narrative puzzles solved collaboratively by individuals or groups of people, through location-based games using GPS and phone networks to manipulate game pieces virtually co-located with geographic locations, all the way to a full 3D tracking and pose tracking of users to produce heads-up displays and visual indications of actions taken in a virtual game arena.

Augmented reality systems have been proposed and studied for at least two decades, with varying effect, for use in robotic interfaces. There has been increasing prominence in recent years due to a recent increase in graphics technology and tracking methods. There are two areas within human-robot interaction where augmented reality is used extensively. Robot arm prediction and planning and data visualization and enhancement.

Robotic arms are often built with complex mechanisms having high degrees of freedom, and as such have complex mechanical interactions governing the path of their end effectors and the robot body. Augmented reality has been used to show both the path result of the planning to be used for the robot to reach a goal state, as well as showing where the body of the robot will end up when the arm reaches it's end effect[78]. It has also been used in situations where motion of a robot arm is a critical task that must be performed repeatedly without error in an environment where small changes prevent automated driving of the effectors, such as space station docking[79].

In mobile robotics, the sensor readings or algorithm results cannot be clearly visualized by the operator or programmer when debugging the robot. Collett and MacDonald[80] developed a system where sensor information is overlaid on an augmented display to show the results of sensor data and algorithms which are running on a robot, from an overhead camera or head-mounted display. The path of the robot historically, or the planned path could be displayed and the sensors are drawn in the environment using the current location of the robot as an anchor. Recently, Daily et al.[81] presented a system that uses augmented reality to create an extra layer of information when looking at a

swarm using an augmented display. They overlaid information such as robot movement intentions or robot identity information.

Nunez et al.[82] used AR methods along with accurate localization of a autonomous robot with a scanning laser to produce an AR camera view which shows the sensor data being returned by the robot as well as the detected walls, corners and the topological map used for planning. This is similar to the system proposed here but has significantly higher sensor requirements, and focuses on planned actions instead of previous actions.

In search and rescue robotics, an augmented reality interface was developed[30] fusing sensor information and camera information, placing a virtual avatar of the 3D robot being controlled within an interface. The video from the teleoperated robot was then projected as a sheet in front of the robot, and the previous locations of the robot were also displayed on the map as it was being built. The interface built is a good example of the type of interface that can be created with augmented reality.

Adding augmented reality to a teleoperated view has also been explored recently. Lera et al.[83] used markers placed in the environment to provide navigational aids to robot operators by replacing the markers with virtual arrows in the augmented view. The markers were placed in the environment beforehand but could be used to guide different robots on different paths. In some cases the augmented reality added is to accomplishes the same result as the external camera sensor overlays and shown in a first-person view. In other times, it can be used to show information which is available from other sensors overlaid on the main view to allow the operator to focus on control of the robot.

10.4 Trail Placement and Rendering

The system built uses the localization of the robot in order augment the vision of a camera in a teleoperated robotic system to display the estimated previous path of the robot in the world which is being traversed. There are four stages to accomplishing this task - collecting the path as a localization stream including collecting the location history, placing the augmented trail in the simulated world which is to be viewed by the user, placing the camera within this world so the path is displayed on the screen in the expected location, and overlaying the virtual camera image on the actual image

which is displayed for the user. The four steps are shown in a block diagram form in Figure 10.2. There is also an optional step in the system which enhances the augmented reality if there is a range finder available to the system.

Collecting the localization history is the only part of the system which gets data from the robot. The path is collected over time by logging the localization stream and separating it into waypoints which are separated by distance or time in the track of the robot. The threshold for adding a new point is $dist(x_i, x) > 0.5m$ **or** $(dist(x_i, x) > 0$ **and** $time(x_i, x) > 5s)$, where x_i represents the last recorded waypoint, x represents a candidate new waypoint (usually the current location and time) and *dist* and *time* measure the distance in space and time between two waypoints. This waypoint trail is computed and stored in the client, which can rebuild it as necessary from a location log which is recorded on the robot. The robot's location storage is used to initialize the client on startup, and could be discarded or limited in length it is burdensome for the robot. If no initial history is received, all processing can be performed on the client without modification of the robot connected to. Optionally, the system can also record more data about each waypoint, such as the confidence in the localization or the current readings from sensors.

Given the trail of waypoints collected in the first step of the system, rendering the trail in the simulated world requires two reference frames. The first is the global reference in which the localization trail is placed in, and the second is the current localization of the camera which is to be used to view this system. Once the global frame is established, a cylinder model is used to construct a set of connected cylinders with ends at (x_{i-1}, x_i) for $i = 2, \ldots, n$ forming a 3D historical trail in the world. The diameter of the cylinders is configurable, but a diameter of 2 centimeters was chosen to provide both a noticeable object in the world while balancing for the occlusion caused. The function of the waypoints being added after a either time or distance provides more points in the path when the motion is more precise and therefore more complicated, providing more detail when needed and conserving rendering resources when the path taken was simple. The path object renders as semi-transparent material to allow the operator to see real-world objects "behind" the path, and provide more feedback when the path occludes itself. Excepting the generated path object, the simulated world is empty.

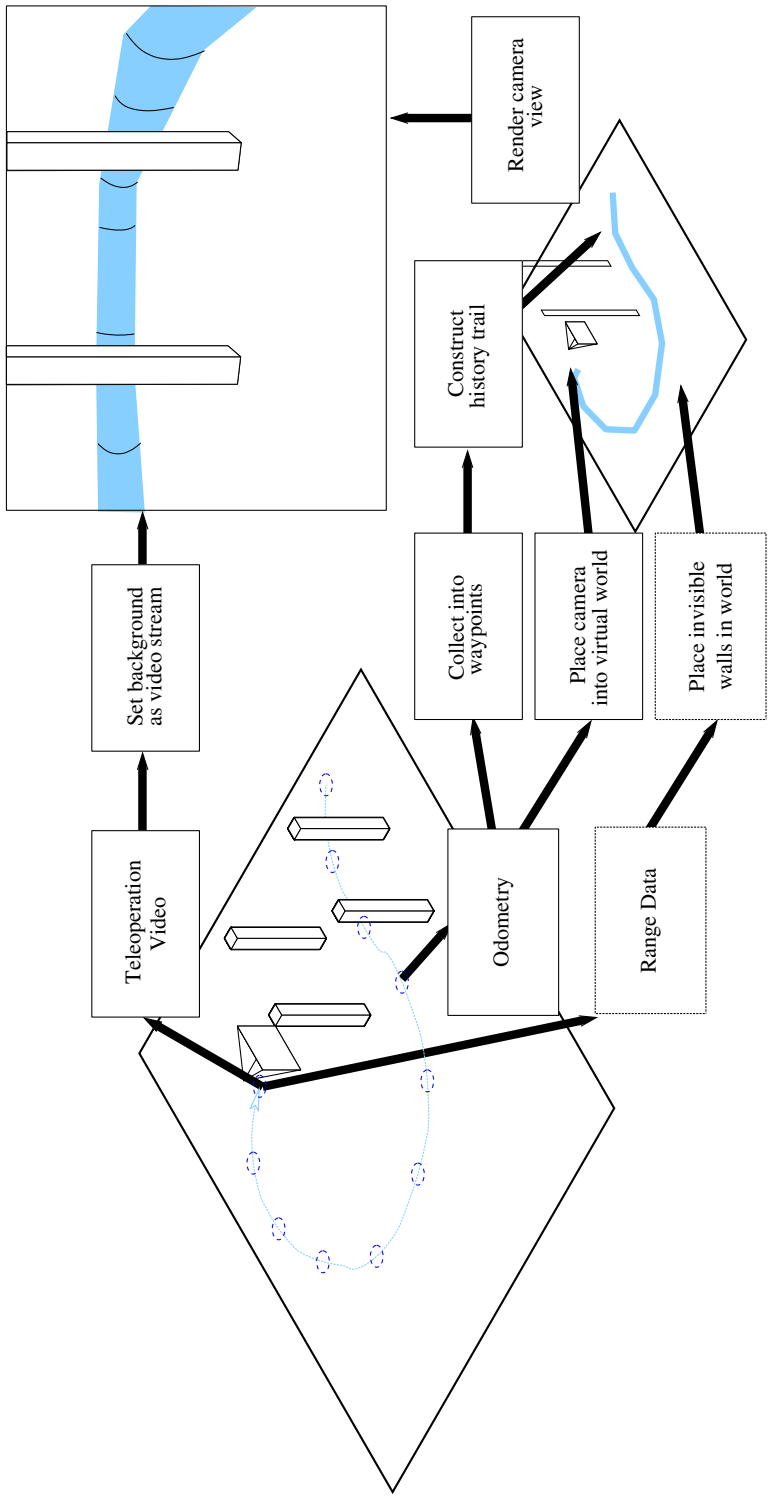


Figure 10.2: Overview of the augmented trail process. Dotted steps are optional for when range data is available.

The rendering camera is placed within the simulated global frame, oriented to match the robot camera's view. To view the real camera view behind the simulated trail, the background render plane of the simulated camera is replaced with video stream images as they are received from the robot middleware. The system renders the camera's view of the simulated world, producing a rendered view including any part of the history trail which is in view. This rendered view is published as an additional camera view to the robot middleware, with the camera pose updating as the robot travels in the environment. The system synchronizes the camera view and localization data to provide a consistent view. If the video or the localization becomes skewed or delayed, the video is prioritized and the trail is not updated temporarily to ensure the operator can smoothly control the robot. In ad-hoc testing, the synchronization of the localization and video succeeded consistently and published rendered images with approximately 30ms lag from input to rendered output. This was sufficient to produce an AR result acceptable for teleoperation.

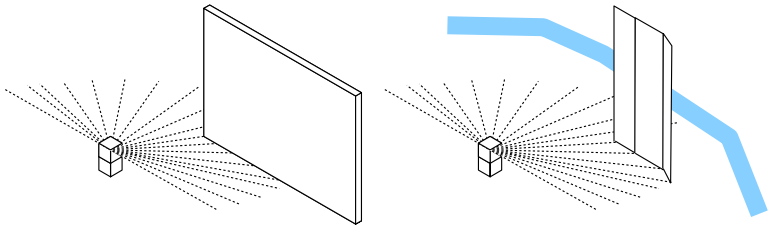


Figure 10.3: Virtual wall placement.

Although the basic system provides the necessary history trail visualization, an optional enhancement was implemented and made available for robots that have sensors beyond the base requirements available. If a ranging sensor of any kind is available, the rendering of the waypoints is enhanced by adding transparent view-blocking walls placed at locations which contain obstacles. Each wall has a height from zero to infinite height. These walls enhance the augmented reality by preventing portions of the history trail which would not be visible if the trail was a real object from being visible. Given a set of two adjacent readings, the local coordinates $(x_i, y_i), (x_{i+1}, y_{i+1})$ are calculated, translated to the global plane, and then a wall is added in between the two readings, as shown in Figure 10.3. If the readings include a confidence, the value must be above a threshold before the wall is placed. Additionally, there is an upper limit to the length

of each wall segment to avoid placing a virtual wall where an opening large enough for the robot might exist.

10.5 Experiment Setup

A set of experimental trials was designed and performed to validate this method of interface enhancement.

The MicroVision mobile robot platform was selected as the system to use in the experiment, as it is highly agile and maneuverable, as well as containing enough computing power on-board to produce an experimental system which only consists of the robot itself and the controller system. The MicroVision contains an Intel Atom processor for control and sensor processing and a microprocessor board to control motors, track the wheel encoders and provide simple localization integrating wheel encoder data and a MEMS Gyroscope and Accelerometer. Each MicroVision has two distance-ranging sensors - a laser range finder with a 270 degree field of view, and a Primesense RGB-D sensor. It also contains ambient light, temperature, and audio sensors, not used in this experiment. The RGB-D sensor and the laser range finder could be used to generate the virtual walls, but were not used in these trials due to hardware failures. A Logitech USB camera was connected and used for the video view. It also contains a 802.11 wireless adapter used to generate a wireless network for clients to connect, and runs a ROS[36] master server for controlling the robot either locally or remotely. A photo of the MicroVision set up for the experiment can be seen in Figure 10.4.

The augmented reality trail system was implemented using the OGRE rendering system[34], used by others to create a number of commercial games as well as debug and development viewers built into ROS and Gazebo. It was integrated as a node in the ROS middleware used for the MicroVision - receiving both odometric data and video and publishing the augmented trail image as an alternate camera. A simple teleoperation system was created using a laptop displaying a configurable video stream from the ROS system and providing teleoperation control using a PlayStation 3 joystick mapped to work using standard ROS teleoperation messages.

A maze was constructed which was unseen by the participant. The maze was constructed of 3 foot cardboard dividers and was therefore reconfigurable between each run

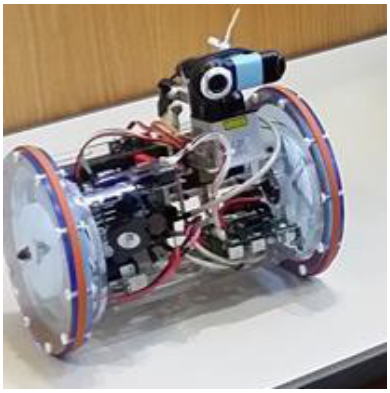


Figure 10.4: MicroVision Robot.

Figure 10.5: Example maze setup.

of the experiment. The maze as constructed was 18 feet on each side, and contained 243 ft^2 of space. A typical setup is shown in Figure 10.5. Inside the maze was placed, at least two meters away from the entrance, three stuffed animals as goal objects. Two different mazes were designed using the same outline, ensuring the same amount of space to explore. The maze layouts can be seen in Figure 10.6. Both mazes are of similar complexity, containing one loop and a number of branching points. Each maze had only one entrance/exit from the perimeter. There were seven visually different animals used; three were randomly chosen at the beginning of each run of the experiment, and placed in the indicated positions. The task communicated to the participant was to explore the maze, finding any objects in the maze, and exit the maze through the entrance as quickly as possible.

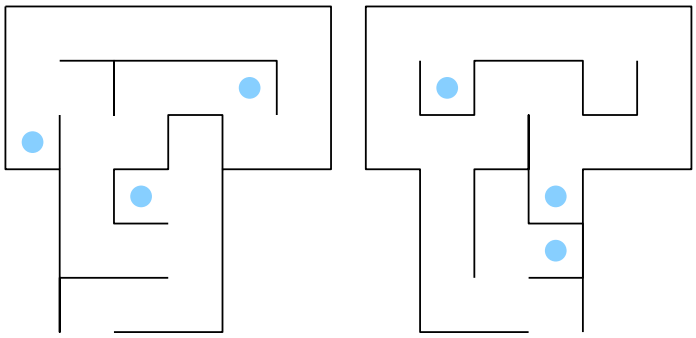


Figure 10.6: The two maze layouts used.

The participant operating the robot was introduced to the experiment task and introduced to the controls. After the participant was comfortable with controlling the robot, the robot placed at the entrance to the maze and the goals of the task were explained. The number of objects placed in the maze was not revealed to the participant before the task started. Each run was timed starting when the robot crossed the threshold of the entrance to the maze, and finishing when passing the same threshold exiting the maze. Both camera feeds, the odometry reported by the robot, and the input from the participant were all recorded. The location of the robot was tracked using both manual review of the video as well as the odometry. After the participant completed the first maze task, a questionnaire was administered with the participant answering open-ended questions about the objects found, and qualitative questions about the ease of use of the robot.

While the participant answered the survey, the maze was changed to the second configuration. The second task was identical to the first task, but with the augmented reality trail enabled. The augmented reality trail visualization was not explained to the participant before entry into the maze. The same post-task survey was administered after the second task was completed. After both tasks were completed, a final survey with questions comparing the tasks qualitatively and collecting demographics was completed. The survey contained questions on their previous experience with video games and robots, and provided free-form space to volunteer any thoughts about the interface or experiment.

Objective measures used in the experiment include the total time the robot was in the maze for each run, and the exploration completeness of the maze, measured in grid squares checked using the camera of the robot. Recall of the animals encountered, including number and type accuracy of recall were included. The path of the robot was reviewed to categorize unnecessary movement, defined as movement where the grid square traversed had been explored previously and was not on the shortest path to the exit. Unnecessary movement was measured in meters traveled according to odometry. The amount of time used in unnecessary movement was also recorded.

Subjective measures from the surveys were compiled to determine preference for each method and collect comments about the methods used. One survey about the maze task was given after each task was completed, and additionally demographics and previous

Questions	Extremes	
How easy of difficult did you find it to control the robot in the maze?	Easy	Difficult
How completely were you able to explore the maze?	Not at all	Completely
How often did you feel that you were confused or lost?	Never	All the time
Would you be able to direct another person to the missing animals?	Every time	Not at all
∗ Did you feel that the trail of the robot was accurate?	Not at all	Completely
∗ Did the trail of the robot block out anything you wanted to see?	Not at all	Definitely

Table 10.2: Scaled response questions. (∗ – only asked after both tasks completed)

experience with robotics and game interfaces was collected. Some question responses were on a seven-point scale, shown in Table 10.2. Open-ended or direct answer questions included the total time the participant believed they were completing the task (for each maze) and whether the trail interface was preferred.

10.6 Experiment Results

Participants were recruited the Minneapolis and Saint Paul metro area using emails to local mailing lists, announcement on social media, and by word of mouth. Each participant was scheduled thirty minutes to complete both tasks and the surveys. All participants finished within the scheduled time. Participants were offered $5 as compensation for their time. Experiments were run in 3 sessions on different days, with 27 participants contributing. The robot was in use for 385 minutes, travelling approximately 1690 meters. No hardware or software failures were experienced.

The participants either experienced the augmented reality view first or second to counterbalance for training effects of slightly more experience with the exact task and robot being controlled. 12 participants completed the augmented reality view task first, and 15 used the non-enhanced view first.

The null hypothesis used states the AR view has no effect on the measured results of the maze task: time taken to complete the maze, exploration percentage of the maze, unnecessary travel distance, and recall would be the same. Each of the four objective

measures were tested independently. Analysis performed on the results is presented here.

Examining first the time to complete the task, the AR view was completed 30 seconds faster than the unaltered view on average, a significant ($t(26) = 1.867, p = 0.036$) difference. It also resulted in approximately half the amount of unnecessary time ($t(26) = 2.08, p = 0.023$), and a reduction in unnecessary distance traveled ($t(26) = 1.9033, p = 0.034$). Almost all runs completely explored the maze, with no significant difference in the number of unexplored squares ($t(26) = 0.8272, p = 0.208$). Most also spotted all three of the animals left in the maze, and recalled them correctly, with no discernible difference between the AR and non-AR views.

Of the 27 participants, 7 preferred the display without the augmented trail, with some commenting on how keeping track of the environment "in their head" was preferred to keep them alert, and others calling the trail "confusing" or "cluttered". Of the 20 who preferred the trail interface, many gave notes indicating the intended purpose of showing the historical trail of the robot was understood.

These results show that the augmented reality trail is a beneficial interface enhancement for teleoperation and successful at indicating where the robot has been and is also preferred by the operators of robots in search and exploration tasks.

10.7 Future Directions

The current system shows only the previous path of the robot and does not add more sensor data onto the display either as an augmented view or using other gaming interactions. The more sparse display was used to simplify the interface for new operators. It would be simple to enhance this system with other sensors or user input to show explored rooms as different colors, allow users to mark significant points with virtual markers, or show the path sections with different colors based on variable interactions, for example time existing causing the trail sections to "fade out" over time.

Only ground robots were tested here, but the method has no inherent planar requirement and could be even more valuable when a third dimension is a factor - the addition of AR trails to hovering-type UAVs such as quad-rotor helicopters could make exploration of complex structures such as caves and multiple-floor buildings easier.

10.8 Summary

Teleoperation is a dominant paradigm of control for military and service robots in the field today. In unmarked environments, wayfinding can be a problem for operators who are unfamiliar with the architecture of the area they are navigating. An enhanced standard teleoperation interface was developed implementing a number of game interactions to produce an easily usable robot control interface. This interface implemented the **maximized environment view** and **joystick movement** interactions as a base. The **history trail** interaction was described, parameterized and implemented using technologies from a graphics engine following the Game Interactions to Robotics framework. This enhancement has very low requirements form the controlled robot, but can use advanced sensing to improve the display if available. A study involving 27 participants using a mobile robot to compare the base interface to the one including the **history trail** interaction was executed, showing significant advantages when the interaction was enabled.

The development and testing of this interface using the methods described in the Game Interface to Robotics framework along with the other three interfaces are examples of using the framework to produce positive changes in robotic interfaces when deriving interactions that are sourced from video games. The augmented history trail specifically shows that it can be used with first-person teleoperation, a very different type of interface but still analogous to the game interfaces which are used. The framework made it simple to identify the portions of the interaction which were important for the robotic interaction and aided in designing the experiment as well as simplifying the interface to make comparison easier.

Chapter 11

Conclusion and Discussion

Gaming interfaces are rich with interactions for navigating avatars, exploring worlds, and performing actions quickly. These interaction designs can be examined and appropriated for use in different contexts, and can relate directly to methods needed to produce a robotic control interface. The interactions which players of games use have a number of advantages including familiarity to a large set of people, commodity availability of hardware, and a large amount of industry and consumer testing, which are highly desirable for robotic control interfaces. To use these interactions, they must be studied for their key properties and translated to equivalent robotic interface elements. A clear path is needed to produce robotic interfaces that exploit the advantages of the gaming interactions.

The Game Interaction to Robotics framework uses a four stage process and fulfills this need. Game interactions that are beneficial to robotic interfaces are identified using heuristics focusing on the resulting robotic interaction goals. Parameterization identifies the different parts of an interaction and guides the translation of the objects, processing, and feedback into a proposed robotic interface. Development of a base interface to compare and the implementation of the result robotic interface proceeds, followed by testing the interface. Guidance including a set of tests of various arduousness, effectiveness, and efficiency using well-known robotic interface metrics are included to show the difference that is present using the new interaction. Along with examples provided, failure can occur if the interaction is not suitable, and can be spotted at each stage of the process using some guidelines.

As the video game industry is relatively mature and only a small number of robotic interfaces derive their interactions from games, there is a variety of interactions which have been used in the video game industry which the framework can immediately be applied to. A survey of 20 recent successful games taken from different genres was conducted and common game interactions were identified. The resulting 26 interactions serve as partial examples of the framework's efficacy in identifying beneficial interactions from many different types of games. Each of these game interactions was scrutinized through the lens of the framework and parameterized using the interaction model. This interaction model was then used to propose enhancements for generic robotic interfaces by mapping the game interaction into the robotic control space.

Throughout the development of the framework, four robotic control interfaces were developed which implemented and tested using a number of the game interactions Each interface was a fully working robotic interface, connecting to a simulator or robot middleware. These interfaces were tested different ways, exercising most of the methods proposed for testing, including experimental trials, remote usability studies, and usability experiments using subjective and objective metrics. Two interfaces included a full example of the framework from start to finish as new game interactions were parameterized, mapped, implemented and tested in the service of developing a single interface. Some interfaces were shown to provide an improvement above the basic interface conclusively. Other interface results were inconclusive quantitatively but the interface improvement was preferred by the participants in the study.

In all cases, the interface improvements required little to no change to the robot platforms being controlled, showing that the game interaction elements are transferable to control improvements on a wide range of mobile robotic platforms. This is a key advantage of transferring game interactions to robotics, as the processing that is normally done by the game engine often can be performed in the robotic client, producing an improved interface for all controlled robots of a type. The wide applicability of improvement of interface intelligence and efficiency in this way can produce an effect across a great number of robot operators.

A remote robotics user study instrumentation framework was developed for use evaluating two of the interfaces developed. The framework reduces the amount of organization and manpower needed to administer a study of a robotic interface where the

robots can be simulated. The ability for the user study to be run on a person's private computer can increase the participation pool. It implements a client-server method to retrieve experimental trial information and submit data relevant to the experiment to a server which tracks the progress of the study, ensuring balanced studies that are also random. The server also acts as a platform for the participants to spread the study through word of mouth on social networks and compare results. Game interaction improvements are well-suited to this framework as they can often be distilled to a simulated robot setting which retains elements from the game the interactions are sourced from, possibly providing a "fun" factor while not compromising the integrity of the study.

Gaming continues to advance the state of art of technology, and as new interface paradigms are introduced, the large amount of capitol being invested in gaming suggests that any number of new technologies will be tested in the lab and in the market in the coming years. The Game Interaction to Robotics framework is easily used on any technology or interface from games, providing a blueprint for testing if technology can be migrated usefully to robotics interfaces. The widespread proliferation of RGB distance sensors from companies such as Primesense migrating from gaming systems into a high density and low-cost sensor for robots is a recent example of a strong effect in this vein. Given the size of the game industry, it is without doubt that games will continue to provide a rich source for new interaction methods and technologies.

Robotics is a field which, sharing a number of goals that game interfaces are targeting, can benefit from these advancements very quickly and without too much change in the elements which are presented. The integration of these new interactions into robotic control interfaces can accelerate the adoption and lessen the training time of the new robot operator as familiar controls and feedback are included in new interfaces. The game interactions included provide benefit to the robot operator in measurable ways that are relevant to robot control. By using the Games Interactions to Robotics framework, these interaction techniques in common use in video games translate to valuable improvements for mobile robotics control interfaces.

References

[1] International Federation of Robotics. World robotics - industrial robotics 2013. http://www.worldrobotics.org/uploads/media/Executive_Summary_WR_2013.pdf, 2013. Last retrieved Nov 26, 2013.

[2] Entertainment Software Association. 2013 essential facts about the computer and video game industry. http://theesa.com/facts/pdfs/ESA_EF_2013.pdf, 2013. Last retrieved Nov 25, 2013.

[3] Joey Davidson. The awkward control design of kid icarus: Uprising. http://www.craveonline.com/gaming/articles/185091-the-awkward-control-design-of-kid-icarus-uprising, March 2012. Last retrieved Nov 26, 2013.

[4] Jeff Hoard. "The Wonderful 101" a Tough Game to Master (review). http://www.theoaklandpress.com/arts-and-entertainment/20130912/the-wonderful-101-a-tough-game-to-master-review/5, September 2013. Last retrieved Nov 26, 2013.

[5] Electronic Arts, Digital Illusions CE. Battlefield 4, 2013.

[6] Roger Caillois. *Man, Play, and Games*. Simon and Schuster, 1961.

[7] Espen Aarseth. Computer game studies, year one. *Game Studies*, 1(1), July 2001.

[8] *Game Developers Conference*, volume 1, San Francisco, California, USA, March 2014.

[9] C. Crawford. *Chris Crawford on Game Design*. New Riders, 2003.

[10] Andrew Rollings and Ernest Adams. *Andrew Rollings and Ernest Adams on Game Design.* New Riders Publishing, May 2003.

[11] Erik Fagerholt and Magnus Lorentzon. Beyond the HUD - user interfaces for increased player immersion in FPS games. Master's thesis, Chalmers University of Technology, 2009.

[12] Andreas Gregeren and Toben Grodal. Embodiment and interface. In Bernard Perron and Mark J. P. Wolf, editors, *The Video Game Theory Reader 2*, chapter 4, pages 65–84. Routledge, New York, New York, 2009.

[13] Chris Klochek and I. Scott MacKenzie. Performance measures of game controllers in a three-dimensional environment. In *Proceedings of Graphics Interface 2006*, GI '06, pages 73–79, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[14] Jakob Nielsen. *Usability Engineering.* Academic Press, Inc., 1993.

[15] Manolya Kavakli and Jason R. Thorne. A usability study of input devices on measuring user performance in computer games. In *Proceedings of the First International Conference on Information Technology and Applications. (ICITA 2002)*, November 2002.

[16] David Pinelle, Nelson Wong, and Tadeusz Stach. Heuristic evaluation for games: Usability principles for video game design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. (CHI '08)*, pages 1453–1462, New York, NY, USA, 2008.

[17] Robin R. Murphy and Debra Schreckenghost. Survey of metrics for human-robot interaction. In *Proceedings of the 8th ACM/IEEE International Conference on Human-robot Interaction. (HRI'13)*, pages 197–198, March 2013.

[18] Aaron Steinfeld, Terrence Fong, David Kaber, Michael Lewis, Jean Scholtz, Alan Schultz, and Michael Goodrich. Common metrics for human-robot interaction. In *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction. (HRI '06)*, pages 33–40, Salt Lake City, Utah, USA, 2006.

[19] Mica R. Endsley. Situation awareness global assessment technique (SAGAT). In *Proceedings of the IEEE 1988 National Aerospace and Electronics Conference. (NAECON 1988)*, pages 789–795 vol.3, 1988.

[20] Dan R. Olsen, Jr., Stephen Bart Wood, and Jonathan Turner. Metrics for human driving of mutliple robots. In *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, pages 2315–2320, New Orleans, LA, April 2004.

[21] P. J. Mitchell, M. L. Cummings, and T. B. Sheridan. Mitigation of human supervisory control wait times through automation strategies. Technical Report HAL2005-02, MIT Humans and Automation Laboratory, 2005.

[22] Sandra G. Hart and Lowell E. Staveland. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In Peter A. Hancock and Najmedin Meshkati, editors, *Human Mental Workload*, pages 139–183. Elsevier, 1988.

[23] M. A. Goodrich, E. R. Boer, J. W. Cr, R. W. Ricks, and M. L. Quigley. Behavioral entropy in human-robot interaction. In *Proceedings of Performance Metrics for Intelligent Systems. (PerMIS'04)*, pages 24–26, August 2004.

[24] Holly A. Yanco, Jill L. Drury, and Jean Scholtz. Beyond usability evaluation: Analysis of human-robot interaction at a major robotics competition. *Human-Computer Interaction*, 19(1):117–149, June 2004.

[25] Michael A. Goodrich and Dan R. Olsen, Jr. Seven principles of efficient human robot interaction. In *Proceedings of the 2003 IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 3942–3498, October 2003.

[26] Marjorie Skubic, Derek Anderson, Samuel Blisard, Dennis Perzanowski, and Alan Schultz. Using a qualitative sketch to control a team of robots. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation. (ICRA 2006)*, pages 3595–3601, May 2006.

[27] Kazuhiko Kawamura, Phongchai Nilas, Kazuhiko Muguruma, Julie A. Adams, and Chen Zhou. An agent-based architecture for an adaptive human-robot interface.

In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences. (HICSS'03)*, January 2003.

[28] Frederick Heckel, Timothy M. Blakely, Michael Dixon, Chris Wilson, and William D. Smart. The WURDE robotics middleware and RIDE multi-robot tele-operation interface. In *AAAI Mobile Robot Competition 2006: Papers from the AAAI Workshop*, pages 10–14, Boston, MA, July 2006. Available in AAAI Technical Report WS-06-15.

[29] Brenden Keyes, Mark MicIre, Jiil L. Drury, and Holly A. Yanco. Improving human-robot interaction through interface evolution. In Disuke Chugo, editor, *Human-Robot Interaction*. Intech, February 2010. Available from http://www.intechopen.com/books/human-robot-interaction/improving-human-robot-interaction-through-interface-evolution.

[30] Curtis W. Nielsen. *Using Augmented Virtuality to Improve Human-Robot Interactions*. PhD thesis, Brigham Young University, February 2006.

[31] M. Waleed Kadous, Raymond Ka-Man Sheh, and Claude Sammut. Effective user interface design for rescue robotics. In *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction. (HRI '06)*, pages 250–257, Salt Lake City, Utah, USA, 2006.

[32] Jijun Wang, Michael Lewis, Stephen Hughes, Mary Koes, and Stegano Carpin. Validating USARSim for use in HRI research. In *Proceedings of the Human Factors and Ergonomics Society 49th Annual Meeting*, pages 457–461, September 2005.

[33] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS 2004)*, volume 3, pages 2149–2154, October 2004.

[34] Torus Knot Software. OGRE - open source 3d graphics engine. `http://www.ogre3d.org/`. Last retreived November 27, 2013.

[35] Russel Smith. Open dynamics engine. `http://www.ode.org/`. Last retreived November 27, 2013.

[36] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software in Robotics*, 2009.

[37] J. Martin Graetz. The origin of Spacewar. *Creative Computing*, August 1981.

[38] Nikolas Engelhard, Felix Endres, Jürgen Hess, Jürgen Sturm, and Wolfram Burgard. Real-time 3d visual slam with a hand-held rgb-d camera. In *Proceedings of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, Vasteras, Sweden, April 2011.

[39] Victoria Woollaston. The sensor that reads your heartbeat from four feet away WITHOUT touching you. `http://www.dailymail.co.uk/sciencetech/article-2408748/The-sensor-reads-heartbeat-feet-away-WITHOUT-touching-you.html`, September 2013. Last retrieved November 28, 2013.

[40] Bei Yuan, Eelke Folmer, and Frederic C. Harris, Jr. Game accessibility: a survey. *Universal Access in the Information Society*, 10(1):81–100, March 2011.

[41] Valve Interactive. Steam controller. `http://store.steampowered.com/livingroom/SteamController/`. Last retrieved Nov 25, 2013.

[42] Ben D. Harper and Kent L. Norman. Improving user satisfaction: The questionnaire for user interaction satisfaction version 5.5. In *Proceedings of the 1st Annual Mid-Atlantic Human Factors Conference*, pages 224–228, 1993.

[43] Paulo G. de Barros, Robert W. Lindeman, and Matthew O. Ward. Enhancing robot teleoperator situation awareness and performance using vibro-tactile and graphical feedback. In *IEEE Symposium on 3D User Interfaces (3DUI)*, pages 47–54, 2011.

[44] Raj M. Ratwani, J. Malcolm McCurry, and J. Gregory Trafton. Single operator, multiple robots: an eye movement based theoretic model of operator situation awareness. In *Proceedings of the 5th ACM/IEEE International Conference on Human-Robot Interaction. (HRI'10)*, pages 235–242, New York, NY, USA, 2010.

[45] Adam Jacoff, Elena Messina, and John Evans. Reference test courses for autonomous mobile robots. In *Unmanned Ground Vehicle Technology III*, volume 4364, pages 341–348, 2001.

[46] J. Bennett Whiteside and K. Holtzblatt. Usability engineering: our experience and evolution. In *Human-Computer Interaction*, 1988.

[47] Holly A. Yanco and Jill Drury. "Where am I?" Acquiring situation awareness using a remote robot platform. In *Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2835–2840, 2004.

[48] Ledinsky Software GmbH. F-sim space shuttle, 2012.

[49] Aerosoft. World of subways 3: London, 2011.

[50] Electronic Arts, BioWare. Mass effect, 2008.

[51] Xun S. Zhou and Stergios I. Roumeliotis. Multi-robot SLAM with unknown initial correspondence: The robot rendezvous case. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS 2006)*, pages 1785–1792, October 2006.

[52] Electronic Arts, Digital Illusions CE. Mirror's edge, 2009.

[53] Blizzard Entertainment. Starcraft II: Wings of Liberty, 2010.

[54] Victor Eruhimov and Wim Meeussen. Outlet detection and pose estimation for robot continuous operation. In *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS 2011)*, pages 2941–2946, 2011.

[55] Blizzard Entertainment. World of Warcraft, 2004.

[56] Sean Targett, Victoria Verlysdonk, Howard J. Hamilton, and Daryl Hepting. A study of user interface modifications in world of warcraft. *Game Studies*, 12(2), December 2012.

[57] Blizzard Entertainment. Diablo III, 2012.

[58] Sierra Entertainment, Valve Corporation. Half life 2: Episode 2, 2007.

[59] Noam Tractinsky. Aesthetics and apparent usability: Empirically assessing cultural and methodological issues. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems. (CHI '97)*, pages 115–122, New York, NY, USA, 1997. ACM.

[60] Susan Drayn and David Siegel. Remote possibilities?: International usability testing at a distance. *interactions*, 11(2):10–17, March 2004.

[61] Morten Sieker Andreasen, Henrik Villemann Nielsen, Simon Ormholt Schrøder, and Jan Stage. What happened to remote usability testing?: An empirical study of three methods. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1405–1414, New York, NY, USA, 2007. ACM.

[62] Anders Bruun, Peter Gull, Lene Hofmeistere, and Jan Stage. Let your users do the testing: A comparison of three remote asynchronous usability testing methods. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1619–1628, New York, NY, USA, 2009. ACM.

[63] Costas S. Tzafestas, Nektaria Palaiologou, and Manthos Alifragis. Virtual and remote robotic laboratory: comparative experimental evaluation. *IEEE Trans. on Education*, 49(3):360–369, Aug 2006.

[64] Andrew Drenner, Michael Janssen, and Nikolaos Papanikolopoulos. Coordinating recharging of large scale robotic teams. In *Proceedings of 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009.*, pages 1357–1362, Oct 2009.

[65] Andrew Drenner, Michael Janssen, and Nikolaos Papanikolopoulos. A strategy for improving observability with mobile robots. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA 2010)*, pages 420–425, May 2010.

[66] Toby H.J. Collett, Bruce A. MacDonald, and Brian P. Gerkey. Player 2.0: Toward a practical robot programming framework. In *Proceedings of the Australasian Conference on Robotics and Automation. (ACRA 2005)*, Sydney, Australia, December 2005.

[67] Bradley Kratochvil, Ian T. Burt, Andrew Drenner, Derek Goerke, Bennett Jackson, Colin McMillen, Christopher Olson, Nikolaos Papanikolopoulos, Adam Pfeifer, Sascha A. Stoeter, Kristen Stubbs, and David Waletzko. Heterogeneous implementation of an adaptive robotic sensing team. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation. (ICRA 2003)*, volume 3, pages 4264–4269, September 2003.

[68] Andrew Drenner, Ian Burt, Tom Dahlin, Bradley Kratochvil, Colin McMillen, Brad Nelson, Nikolaos Papanikolopoulos, Paul E. Rybski, Kristen Stubbs, David Waletzko, and Kemal Berk Yesin. Mobility enhancements to the scout robot platform. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation. (ICRA 2002)*, volume 1, pages 1069–1074, May 2002.

[69] C. I. Connolly, J. B. Burns, and R. Weiss. Path planning using laplace's equation. In *Proceedings of 1990 IEEE International Conference on Robotics and Automation. (ICRA 1990)*, volume 3, pages 2102–2106, May 1990.

[70] Michael A. Goodrich, Timothy W. McLain, Jevverey D. Anderson, Jisang Sun, and Jacob W. Crandall. Managing autonomy in robot teams: observations from four experiments. In *Proceedings of the 2nd ACM/IEEE International Conference on Human-robot Interaction. (HRI'07)*, pages 25–32, 2007.

[71] Richard T. Vaughan. Massively multi-robot simulations in stage. *Swarm Intelligence*, 2(2-4):189–208, December 2008.

[72] Jean-Claude Latombe. *Robot motion planning*. Kluwer Academic Publishers, 1991.

[73] Brian F. Goldiez. *Techniques for Assessing and Improving Performance in Navigation and Wayfinding Using Mobile Augmented Reality*. PhD thesis, University of Central Florida, 2004.

[74] Ralf Mayet, Jonathan Roberz, Thomas Schmickl, and Karl Crailsheim. Antbots: A feasible visual emulation of pheromone trails for swarm robots. In Marco Dorigo, Mauro Birattari, Gianni A. Caro, René Doursat, Andries P. Engelbrecht, Dario Floreano, Luca Maria Gambardella, Roderich Groß, Erol Şahin, Hiroki Sayama, and Thomas Stützle, editors, *Swarm Intelligence*, volume 6234 of *Lecture Notes in Computer Science*, pages 84–94. Springer Berlin Heidelberg, 2010.

[75] Hongen Liao, Hirotaka Ishihara, HuyHoang Tran, Ken Masamune, Ichiro Sakuma, and Takeyoshi Dohi. Fusion of laser guidance and 3-d autostereoscopic image overlay for precision-guided surgery. In Takeyoshi Dohi, Ichiro Sakuma, and Hongen Liao, editors, *Medical Imaging and Augmented Reality*, volume 5128 of *Lecture Notes in Computer Science*, pages 367–376. Springer Berlin Heidelberg, 2008.

[76] The Astonishing Tribe (TAT). TAT Augmented ID. `http://www.youtube.com/watch?v=tb0pMeg1UN0`, 2009.

[77] D. W. F. van Krevelen and R. Poelman. A survey of augmented reality technologies, applications and limitations. *The International Journal of Virtual Reality*, 9(2):1–20, June 2010.

[78] H. C. Fang, S. K. Ong, and A. Y. C. Nee. Robot path and end-effector orientation planning using augmented reality. In *Proceedings of the 45th CIRP Conference on Manufacturing Systems. (CMS 2002)*, pages 191–196, 2012.

[79] Keshav Cintamani, R. Darin Ellis, Chin-An Tan, and Abhilash Pandya. Automated augmented reality operator aids for space robotic teleoperations. In *Human Factors: a view from an integrative perspective. Proceedings of the Human Factors and Ergonomics Society Europe Chapter Annual Meeting*, pages 17–28, Toulouse, France, October 2012.

[80] Toby H. J. Collett and Bruce A. MacDonald. Augmented reality visualisation for player. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation. (ICRA 2006)*, pages 3954–3959, May 2006.

[81] Mike Daily, Youngkwan Cho, Kevin Martin, and Dave Payton. World embedded interfaces for human-robot interaction. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences. (HICSS 2003)*, 2003.

[82] R. Núñez, J. P. Bandera, J. M. Pérez-Lorenzo, and F. Sandoval. A human-robot interaction system for navigation supervision based on augmented reality. In *Proceedings of the 2006 IEEE Mediterranean Electrotechnical Conference. (MELECON 2006)*, pages 441–444, May 2006.

[83] Francisco J. Lera, Camino Fernández Juan Felipe García, and Vincente Matellán. Augmented reality to improve teleoperation of mobile robots. In *Proceedings of the XII Workshop of Physical Agents. (WAF 2011)*, September 2011.